EI258[illegible]45US

SPECIFICATION

To all whom it may concern:

Be it known that John K. Thomasson and Myron L. Mosbarger, citizens of the United States of America, have invented a new and useful invention entitled METHOD AND SYSTEM FOR ASYMMETRIC SATELLITE COMMUNICATIONS FOR LOCAL AREA NETWORKS of which the following comprises a complete specification.

METHOD AND SYSTEM FOR ASYMMETRIC SATELLITE COMMUNICATIONS FOR LOCAL AREA NETWORKS

Ins. A₁

~~**Software Appendix**. This specification includes a software source code appendix which includes the computer source code of one preferred embodiment of the invention. In other embodiments of the invention, the inventive concept may be implemented in other computer code, in dedicated electronic hardware, in a combination of these, or otherwise. This software appendix is hereby incorporated in this application in its entirety and is to be considered to be part of the disclosure of this specification.~~

I. BACKGROUND OF THE INVENTION

Ins A₂

A. Field of the Invention.

This invention relates to methods and systems for communications between computers and other digital information devices. More particularly, this invention relates to communications between computers making use of digital satellite communications channels and computer local area networks, to provide access to the internet, to facilitate data and software distribution, and/or to enhance the capabilities of intranet systems for computers with connections to local area networks.

B. Description of Related Art.

It is well established that computers can communicate across local or wide area networks. It is also well known that satellite receivers and transmitters can be used to transfer high volumes of digital data. Some efforts have been made to provide communication systems which can be used to transfer data between computer processors using a variety of communication mediums (see Moura et al., U.S. Patent No. 5,586,121). However, it is desirable to provide a high-speed, low-cost, satellite-based communication system which is designed to optimize the use of digital satellite systems for local area networks (LANs). Optimizing the use of the digital satellite channel is best accomplished through the use of asymmetrical communications between the computer server and the internet as opposed to symmetric communication, in which substantially the same data rates and the same media are used for both the transmit direction and the receive direction, and as opposed to communication systems which employ asymmetrical communication between the local area network and the server. Particulary, asymmetrical systems ~~which~~ require upstream router hardware, "backbone" network hardware, or dial-up internet service providers (ISPs) to create a "hybrid" asymmetrical local system with a symmetrical local area network. Since calls to the internet can efficiently be made at relatively low speeds, and since using digital satellites as a communication medium provides the capability of very high speed responses from the internet, an asymmetric transmission from the internet across the digital satellite to the LAN server provides the greatest system efficiency.

The most common method of sending and receiving computer information today is a land line service (i.e., a switched service, a dedicated line, and/or an analog modem, each using telephone wire lines). However, such a system encounters may problems, including slow transmission speeds, high access costs, lack of available wire lines, and internet congestion.

Satellite communication receivers are commonly used to create or supplement existing private wide area data and video networks. When used as an extension to a data network, these satellite links may interconnect local area networks. Satellite links can provide many advantages over land line service, including potentially high speed data transmission and wide availability. However, typical satellite links have required expensive hardware both to transmit and to receive data. The expense of the hardware has made the use of satellite communication channels generally unavailable to those who most need it.

This invention addresses these issues by providing a method and system for providing the advantages of satellite communications for high volume download data packets and typically using a relatively low speed land line for the low volume upload data request packets. By capitalizing on the asymmetrical nature of internet dataflow, this invention provides an efficient solution for LAN to satellite internet communications.

For general background material the reader is directed to U.S. Patent Nos. 5,095,480, 5,379,296, 5,423,002, 5,488,412, 5,534,913, 5,539,736, 5,541,911, 5,541,927, 5,555,244, 5,583,997, 5,586,121, 5,594,872, 5,610,910, 5,610,920, 5,631,907, 5,659.692, 5,668,857, 5,673,265, each of which is hereby incorporated by reference in its entirety for the material disclosed therein.

II. SUMMARY OF THE INVENTION

This invention is a method and system for efficiently communicating between networked computers using a high speed satellite communications channel. It is an object of this invention to provide a high speed satellite based information delivery system for local area network connectivity to the internet, for file, data, software, and/or multimedia distribution.

It is a further object of this invention to provide a data transmission system particularly well suited to remote location and/or locations where access to high speed data mediums is unavailable or prohibitively expensive.

It is a further object of this invention to provide a high speed data transmission system that utilizes a highly flexible and adaptable software method.

It is a further object of this invention to provide a high speed data transmission system that communicates with the internet while being internet service provider (ISP) independent.

It is a further object of this invention to provide a high speed data transmission system that makes use of digital satellite communications technology to enhance data bandwidth, channel reliability, and accessability.

It is a further object of this invention to provide a high speed data transmission system that utilizes a software method capable of operating on a wide range of server operating systems, including Windows 95, Windows NT, NetWare, Linus, Macintosh, present and future versions and the equivalents.

It is a still further object of this invention to provide a high speed data transmission system that is compatible with a wide range of communication protocols and/or mediums, including ISDN, T1, modem, dedicated phone line, switched phone line, frame relay and ATM.

It is another object of this invention to provide a method for permitting many client computer systems, which may be operating system independent and operating on one or more local area networks, to communicate over a single satellite dish, at very high data rates.

It is a further object of this invention to provide a method using software which can operate on a wide variety of hardware, operating system, and software platforms, including, but not limited to: Macintosh, Linux, Unix, OS/2, and Windows NT.

These and other objects of this invention are readily apparent to individuals of ordinary skill in the art upon further study of the drawings, detailed description, claims and abstract that are included in this patent disclosure.

III. BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 depicts a top level rendering of the major component parts of the communication system invention.

Figure 2 depicts a preferred embodiment of the architecture of the invention.

Figure 3 depicts the preferred flow of data through the several protocol transitions in a preferred embodiment of the invention.

Figure 4 depicts a top level flow diagram showing the primary steps of the process flow for an example single received data packet in the preferred method of the invention.

Figure 5 depicts a detailed flow diagram showing the package delivery major step of the preferred embodiment of the invention.

Figure 6 depicts a detailed flow diagram showing the internet protocol (IP) major step of the preferred embodiment of the invention.

Figure 7 depicts a top level flow diagram showing the primary steps of the process flow for an example transmission of internet protocol datagrams.

Figure 8 depicts additional detail showing the transfer queue (TxQ) thread processing of the filter queue step of the transmission portion of the invention.

Figure 9 depicts additional detail showing new queue (NewQ) thread processing of the filter queue step of the transmission portion of the invention.

Figure 10 provides an example embodiment of the user interface of the invention.

IV. DETAILED DESCRIPTION OF THE INVENTION

This invention is a system and method for asymmetric communications, between a remote information provider and a client computer system residing on a local area network (LAN), using high bandwidth digital satellite communication channels. The preferred embodiment of the method of the invention is performed in software residing on a local computer system. The current preferred embodiment of the method software ~~in~~ is written in Intel 386 assembly code, C and C++ computer languages. The reader is directed to the ~~appended~~ computer software microfiche appendix for a complete disclosure of the software making up the current best mode of the method of this invention. Alternatively, those of ordinary skill in the art could practice this method in a wide variety of procedures, computer languages, or even in dedicated electronic hardware. Therefore this patent should not be read to be limited to the specific embodiment of the provided software microfiche appendix. Rather, this software source code is provided to fully describe one preferred embodiment of the method of this invention. Also, in its preferred embodiment, this invention performs in association with DirectPC satellite receivers, Novell NetWare network software, and standard off-the-shelf computer hardware. Other alternative satellite receiver systems, networking software and computer hardware could easily be substituted by those of ordinary skill in the art without departing from the essence of this invention. Similarly, the preferred embodiment, described in the following detailed description, includes a number of components and method steps which may not be absolutely necessary in other embodiments of the invention. The reader is, therefore, directed to the claims for a description of the range of this patent.

Figure 1 depicts a top level rendering of the major components and communication paths constituting the system 100 of this invention. A client computer 101a-g is shown clustered with other client computers on a local area network (LAN) 102. The client computer 101 uses standard off-the-shelf commercially available software, while the server provides the user interface to the desired information. The LAN 102 provides the means for communicating from the client computer 101 to a server 103, which provides the communication interface outside the LAN. The server 103 receives data from a digital satellite receiver 110, depicted here as a satellite dish, across a signal antenna waveguide 115. The digital satellite receiver 110 receives the digital information from a downlink channel 111, which is transmitted from a geosynchronous satellite 112. The satellite 112 receives the information from a network operations center 114 via an uplink channel 113. The server 103 generally uses the above described communications channel from the network operations center 114 for downloads of information, such as internet web page data, software updates, data file distributions and other similar data packages. Alternatively and in addition, this invention provides the capability of using another satellite communication channel to send requests from the client computer 101 to the network operations center 114. This is a particularly useful feature for access from remote locations. Use of the satellite communication channel provides important benefits to the information requestor at the client computer 101, including very high speed data transfer, the ability to receive broadcast software distributions which in turn means the requestor is likely to receive such distributions in a more timely and cost effective manner, and the ability to have internet access from locations where wired communications channels, such as telephone lines, are either unavailable, overly burdened, or prohibitively expensive.

Figure 1 also shows the preferred, and more typical request communications channel. In this preferred request channel the client computer 101, connected through the LAN 102, through the server 103, sends a request via modem 105, which typically is connected to the server via a standard serial RS-232 cable 104. The modem 105 in turn is connected to standard telephone land lines 107 via a standard phone cable 106. The request is passed across the land lines 107 to the internet service provider 108, which communicates to the internet 109.

This invention is designed to be highly flexible and adaptable to different client computer 101 configurations, both hardware and software as well as with and to a wide variety of communication interfaces. Computer hardware such as personal computers, workstations, mini computers, mainframe computers and special purpose computational equipment can be functional client computers 101 as intended within this patent specification. Similarly, computer system operating systems which are supported and used in the preferred and alternative embodiments of this invention included but are not limited to: Windows 3.1, Windows 95, Windows NT, Macintosh, Linux, Unix, OS/2, NetWare, their current versions, past versions, and equivalent future versions and the equivalent. Communications interfaces that are or can alternatively be used with or as a part of this invention include routers, ethernet, ISDN equipment, switched 56, T1, Token Ring, frame relay, modems, satellite and the equivalent.

The advantage of this preferred mode of operation is that the communication channels are used in the most efficient manner. Typically, request packets are relatively small and can be transferred with minimal impact across land lines. While downloaded packets can be very large with significant amounts of highly concentrated graphics. For the vast majority of client computer 101 users the limitation of internet or the ability to receive other downloaded file information is the time it takes for the download transfer to be accomplished. This problem is solved by transferring the potentially very large downloaded packets (files, graphics and other information) using the high bandwidth satellite channel.

Figure 2 depicts a preferred embodiment of the software architecture of the invention. As shown, in its preferred embodiment, this software operates in association with several standard commercially available software packages and protocols, including Novell NetWare, Ethernet, Token Ring, TCP/IP, IPX and AIO. This software method of the invention also makes use of certain commercially available hardware component, as shown in figure 2, including: the satellite receiver 110, a network router 205 and a modem 104. The reader should understand that this figure 2 presents a single simplified embodiment of the invention. Alternative embodiments could use alternative software packages and protocols, as well as different or multiple hardware components. Figure 2 also shows a single embodiment of the path of information through the various software and hardware components. Download information packets are received by the satellite receiver 110, which in turn communicates electronically with the DPC LAN 210, a commercially available hardware driver. The information next passes through the LSL NLM 202, a routine commercially provided by Novell Incorporated which acts as an intermediary between the driver and the protocol stack to control information packet transfer. Next, the TCP/IP 203 protocol stack receives the information packet. The TCP/IP 203 protocol stack is capable of communicating alternatively with a modem 104, via the LSL NLM 202; the DPCAGENT 207 routine, core to this invention; the AIO 208, a Novell product for managing serial communications; and through the AIOCOMX 209, which is the asynchronious input/output interface to the client computer hardware communication ports, or with router 205, via the LSL NLM 202, the DPCAGENT 207, the LSL NLM 202 and an ethernet driver 204. Alternative embodiments of this invention may make use of other standard commercially available communication protocols, drivers, hardware and software.

Figure 3 depicts the flow of information in the preferred embodiment of the invention from the satellite 112 to the client computer 101 as well as the flow of information back to the internet 109 from the client computer 101. Data is received 301 from the satellite 112 by the satellite receiver 110. Next, the data is transmitted to and received by the server 103 hardware 302 where it is placed in on-board memory. The DPC.LAN DirecPC network card driver retrieves the data packet from hardware memory 303. Next, if the packet is identified as an internet protocol (IP) format packet it is delivered to the IP protocol stack 304. If the packet is identified as a transmission control protocol (TCP) segment, it is delivered to the TCP protocol stack 305. TCP delivers the data packet to a proxy gateway 306. The proxy gateway forwards the data packet to the client computer via the local area network and standard LAN protocol controllers 307. Next, the client computer processes the data and generates a return packet 308. The return packet is delivered to the proxy gateway via the local area network and the standard LAN protocol controllers 309. The return packet is forwarded to the TCP stack 310, and next to the IP stack 311. The IP delivers the return packet to the DPCAGENT.NLM process 312, which delivers the return packet data to a transmit device, such as a modem or a router 313.

Figure 4 depicts a top level flow chart rendering of the major steps of the process flow for a single downloaded data packet section of the invention. Initially the downloaded packet is received 410 by the DirecPC circuit card. This circuit card next transfers the packet data to the DPC.LAN routine 402. In the preferred embodiment of the invention, as shown in the source code appendix, the DPC.LAN routine is denoted as DriverISR proc. This process includes the steps of setting up the RAM adapters and establishing a timestamp for the packet. Further detailed information on the functioning of this routine is found within the software appendix.

Next, a test is made 403 as to whether the received data is a package delivery or an internet delivery. If the received data is package data, it is delivered 404. Package data delivery includes and provides the capabilities of simultaneously broadcasting software upgrades or data files to many client computers, potentially throughout any one continent. Client computers can also request the package data delivery service to retrieve a package of information through the client accessible interface of the invention. If the received data is internet data, then internet data delivery is made 405. Additional detail on steps 404 and 405 follows in this specification.

Figure 5 depicts a detailed flow chart of the preferred embodiment of the package delivery major step of the method of the invention. The data packet is transferred to the DPCAGENT,NLM process 501. This process is a NetWare Loadable Module (NLM) process running on NetWare. After the data packet is received 501, a test is made to determine whether the packet will update the catalog 502. If the catalog will be updated, then it is updated 503 using off-the-shelf commercially available software and the process of package delivery for that packet ends 511. If, however, the catalog will not be updated, then a test is performed to determine whether the site will be updated by the data packet 504. Site updates include modification of such site parameters as NOC versioning, encryption key updates, and becoming a member of a group or leaving a group. If the site will be upgraded, then the process performs the upgrade of the site parameters 505, and the process for this packet ends 511. If the site will not be updated, then the package file is found and stored on the server disk 506. A test is then made to determine whether the end-of-file has been encountered 507. If the end-of-file has not been encountered then the process for that packet ends 511. However, if the end-of-file has been encountered, then a test is made to determine whether there are any "holes" in the file, that is whether the file is incomplete 508. If no holes are found in the file, it is marked as complete 509 and the process for this packet ends 511. If "holes" are found in the file, then a request for partial retransmit of the missing packet is sent 510, at which point the process for this packet ends 511.

Figure 6 depicts a detailed flow chart of the preferred embodiment of the internet protocol delivery major step of the method of the invention. Internet package delivery or Internet Protocol (IP) delivery is a major function of the invention providing the capability of receiving large files from an internet source at a very high speed. First the data packet is transferred to the DPCAGENT.NLM routine 601. A test is made to determine whether the data is in transmission control protocol (TCP) 602. If the data is not in TCP protocol then the data packet is transfered to the Internet Protocol (IP) stack 609 and the process for this data packet ends 610. If the data is in TCP form then a test is made to determine if a "SYN" or beginning of section is being initiated 603. If no "SYN" is detected, then a test is made to determine if an end of session, commonly a FIN or RST command, has been encountered 605. If no such end of session is found, then the data packet is transferred to the IP stack 609 and the process for this packet is ended 610. If, however, a "SYN" is detected, then the inquiry is made as to whether a connection slot is available 604. Connection slots perform the function of managing the number of subscribers permitted to have access to the communication network at a given time. If a connection slot is available, it means that the customer still has client computer access capacity. If a connection slot is available, a connection slot is allocated 607 and the data packet is transferred to the IP stack 609 and the process ends. If it is determined that a connection slot is not available, then the data packet is dropped or discarded 606 and the process for this packet ends 610.

Figure 7 depicts a top level flow chart showing the primary steps of the preferred embodiment of the transmission of internet protocol datagrams (or packets) method steps of the invention. Packets that are received from the IP stack are stored on the NewQ 701. Next the packet is removed from the NewQ and tested against the each and every packet on the TxQ to determine if any TxQ data is redundant or dated and should be replaced 702. If a comparison of the packet with the TxQ packets finds the updated or "newer" information, then the TxQ packet data is replaced by the current packet data. This approach is essential to maintaining the fairness of the TxQ packet transfers while ensuring that good data is transmitted thereby improving the transmission efficiency of the system. A test is performed to determine if the packet was included in the TxQ 703. If the packet was not included then the current or NewQ head packet is dropped or discarded 704. Otherwise, if the packet was included, a test is performed to determine if the NewQ is empty 705. If the NewQ is not empty, the process returns to the test NewQ step where a new NewQ head packet is compared against the TxQ. If, however, the NewQ is empty, then the process enters a wait state 706 where a trigger, that is meeting a specified condition, such as new packet on the NewQ or exit command, is required before the process restarts at the testing step 702.

Figure 8 depicts a detailed flow chart showing the transfer queue (TxQ) thread processing steps of the transmission portion of the invention. In the preferred embodiment of the invention processing the TxQ and processing the NewQ are independent threads of the program which are capable of running independently on one or more computer processors. In processing the TxQ it is first determined whether the TxQ is empty 801. If the TxQ is empty, then the process enters a wait state 805 where a trigger, such as a polling timer, a transmission complete signal, or an exit command, is required to resume processing. Note that in the preferred embodiment of the invention the expected wait time is calculated in this step and the polling time is initiated. If the TxQ is not empty, a test is made to determine if the head packet of TxQ is too old 802. In the preferred embodiment of the invention, too old is defined as a packet that has been in the TxQ for more than sixty (60) seconds. Alternative embodiments could employ any practicable time period. If the TxQ head packet is too old, then it is discarded 806 and the process returns to the TxQ empty test 801. If the TxQ head packet is not too old, then a test is made to determine if the media, or communication conduit, is capable of transferring another packet of data 803. If the media is capable of transferring another packet, then the packet is written to the transmission device 804, otherwise, the process enters the wait state 805 and waits for a trigger as described above.

Figure 9 depicts a detailed flow chart showing the new queue (NewQ) thread processing steps of the filter queue step of the transmission portion of the invention. The filter queue processing step of the invention, which is the core of step 702, is important in providing the communication efficiency which is one of the key objectives of this invention. A test is made to determine whether the NewQ is empty 901. If the NewQ is empty then a wait state is entered 902 where a trigger, such as new packet available in NewQ, exit or timer count, is required to resume the process. If NewQ is not empty, then the head of NewQ is renamed as ECB 903, a packet holding variable. The maximum age of the ECB packet is set 904 and a test is performed to determine if the ECB packet is fragmented 905, that is whether ECB is only a partial packet, which in the current best mode of the invention is not inspected. If ECB is fragmented, then it is appended 906 to the TxQ for transmission. If ECB is not fragmented, then a test is performed to determine if it is a TCP packet 907. If it is not a TCP packet, then the test is made to determine if it is a UDP packet 980. If it is not a UDP packet or a TCP packet then it is appended to the TxQ for transmission 906. If, however, it is a UDP packet, a test is made to determine if ECB is a duplicate of a DNS request 909. If ECB is not a duplicate of a DNS request, then the packet is appended to TxQ 906. If ECB is a duplicate of the DNS request, then it is discarded 910 and the process returns to testing to see if NewQ is empty 901. If it is a TCP packet, then a test is made to determine if ECB is a "SYN" or beginning of session packet 911. If ECB is a "SYN" packet, then a test of whether a connection slot is available is made 912. If a connection slot is available it is allocated 913. If, however, a connection slot is unavailable ECB is discarded 910. If ECB is not a "SYN" packet then the test is made to determine if ECB is a session abort packet (RST) 914. If ECB is found to be a session abort packet TxQ is cleared of all matching packets 915 and a test is made to determine if ECB will terminate a session 916. If ECB is not a session abort then the step of clearing all matching packets 915 is skipped. If the ECB is found to terminate a session, then the a connection slot is released 917. A test is then performed to determine if ECB is void of user data 918. If it contains user data then it is appended to TxQ 906. Otherwise, the End-of-Window-Position (EOWP) is computed 919 and a test is made to determine if ECB's EOWP is greater than the matched TxQ packets 920, if not ECB is discarded 910, and if so, ECB has its EOWP stored in matched TxQ 921 and then ECB is discarded 910.

A Figures 10~~a-l~~a-K show various screen shots demonstrating the user interface of the invention. Figure 10a shows the DCPAGENT routine options for user selection as well as the digital package delivery queue screen. Figure 10b shows a package of data in transit as displayed on the user screen. Figure 10c shows the main screen of the package delivery interface. Figure 10d shows package statistics as displayed for user information. Figure 10e shows configuration control screens where the user can modify certain modem, package delivery and provider configuration information. Figure 10f shows the package delivery configuration editor screens with the information that can be user modified. Figure 10g shows the login script editor and the provider configuration editor. Figure 10h shows additional provider configuration editor screens showing the configuration of an outbound protocol case. Figure 10i shows the dish or antenna pointing adjustments screens. Figure 10j shows the satellite dish signal strength meter for dish alignment. Figure 10k shows the adapter information screen, here showing site information including the card hardware serial number, the site identification, the status of keys, and a list of communities or groups in which the user is participating.

COMPUTER SOFTWARE APPENDIX

~~The following computer software appendix is being deposited as part of the specification of this patent application under 37 C.F.R. 1.96 as original copies from computer printout and as two sheets per one patent specification page for the ease of the publication office.~~

```
#include "dpcagent.h"            /* Our header file */


/*****************************************************************
 *
 * History:
 *
 * Version 0.1 - First Package delivery for demo use.
 * Version 0.2 - (03-15-96)
 *                                                   Added Community/
Site ID filtering to catalog
 *                                                   Added Periodic d
elivery support
 *                                                   Added DPCGetMLID
Stats()
 *                                                   Broke modem conf
ig into file and added DPCUpdateConfig()
 * Version 0.3 - (03-25-96)
 *                                                   Added SFX_PUSH s
upport for interactivity field in
 *                                                   catalog.
 *                                                   Fixed FreeFreeFr
ee... in catalog by adding minimum
 *                                                   length check bef
ore commiting entry.
 * Version 0.4 - (03-29-96)
 *                                                   Added baud rate
indexes to modem configuration and
 *                                                   defaulted them t
o 2400 to fix key reception bugs.
 *                                                   Also added code
to enable agent to continue getting
 *                                                   keys until succe
ssful, even if agent shuts down.
 * Version 0.5 - (04-09-96)
 *                                                   Fixed push files
 showing up in Cancel Download list
 *                                                   Got DisplayMLIDS
tatistics halfway running(doesn't abend)
 *                                                   Recognized cost(
explicit) files in catalogue
 *                                                   Got explicit req
uest/response/confirm code running but
 *                                                   NOC doesn't send
 it yet.
 * Version 0.6 - (04-10-96)
 *                                                   Finished impleme
nting Get MLID stats
 *                                                   Added DPCGetSign
alStrength()
 * Version 0.7 - (04-11-96)
 *                                                   Added Modem Conf
iguration Editor
 *                                                   Added Signal Str
ength Meter
 * Version 0.8 - (04-12-96)
 *                                                   Fixed DPCGetSign
alStrength to report 0 when not connected
 *                                                   Finished impleme
nting cost(explicit) file download
 *                                                   Modified the way
 View Database Entries option worked
 *                                                   so that it shows
 entire entry and is more usable.
 * Version 0.9 - (04-16-96)
 *                                                   Completed Turbo
```

```
Internet support. Can't totally test
 *                                                   yet until IJ gat
eway is up and running.
 * Version 0.10 - (04-24-96)
 *                                                   Added ARP loopba
ck
 *                                                   Fixed missing "e
lse" in IPSendRoutine after call
 *                                                   to GatherFragmen
ts. Turbo Internet works 90%.
 * Version 0.11 - (04-25-96)
 *                                                   Fixed NWSAppennd
AppendStringField in DLO to include
 *                                                   character sets i
nstead of NULL.
 * Version 0.12 - (05-16-96)
 *                                                   Completion of Ph
ase 2.
 *                                                   Places [illegible]: in
 between ATDT and 1-800...
 *                                                   Added su[illegible]t fo
r commas in phone numbers and prefix
 *                                                   Allowed init str
ing to accept & and -
 *                                                   Modified IP and
Gateway strings to have (ISP) follow it
 *                                                   Debug screen dis
played on if -DEBUG is on command line
 * Version 1.00 - (06-11-96)
 *                                                   Added connect ba
ud rate to Modem Status string
 *                                                   Fixed Tx LED so
it doesn't stay in during internet traffic
 *                                                   Added Adapter In
formation screen
 *                                                   Fixed phone numb
er edit strings to allow alphas
 *                                                   Bumped AIO write
 buffer to 4K(default was 1K)
 * Version 1.01 - (06-17-96)
 *                                                   Broke out LAN dr
iver comm code into driverio.c
 *                                                   Added Agent Regi
ster call to DPC.LAN so that it could
 *                                                   call us when it
removes which would allow us to prevent
 *                                                   abends.
 * Version 1.02 - (06-25-96)
 *                                                   Added -NOPD swit
ch to disable package delivery for debug
 * Version 1.04 - (06-25-96)
 *                                                   Revised send rou
tine to send directly to AIO.
 * Version 1.05 - (06-28-96)
 *                                                   Changed serial n
umber in menu to decimal
 *                                                   Added AIO buffer
 size field to modem configuration
 *                                                   Added auto login
 which changed configuration screens
 * Version 1.06 - (07-05-96)
 *                                                   Completely rewro
te tinet fragment routines(jkt).
 *                                                   Added Modem Cont
rol menu.
 *                                                   Added GNU condit
```

```
ionals.
 *                                                                 Added login scri
pt timeouts
 *                                                                 Broke out PPP ro
utines into PPP.C
 *
 *****************************************************************************/

#define AGENT_VERSION   InxMSG("1.20 ", 173)

/**************************************************
 * Global variable used by DPCAGENT.C
 **************************************************/

/*
 * Resource Tag variables.
 */
struct LoadDefinitionStructure *NLMHandle = 0;
struct ResourceTagStructure *allocRTag = 0;
struct ResourceTagStructure *timerTag = 0;
struct ResourceTagStructure *AESTag = 0;
struct ResourceTagStructure *asyncIOTag = 0;
/*
 * NUT and screen ID variables.
 */
NUTInfo                  *NUTHandle = NULL;                          /* main
NUT handle                         */

#if DEBUG_ALL
struct ScreenStruct *DebugScreenID = 0;        /* for OutputToScreen
*/
#endif
WORD                          ScreenHeight = 0;
/* Screen Height(25)                        */
WORD                          ScreenWidth = 0;
/* Screen Width(80)                         */
LONG                          BackgroundPortal = 0;                  /* NUT B
ackground Portal              */
/*
 * Process and thread variables.
 */
BYTE                          **NLMMessageTable = (0);               /* OS Po
inter to messages             */
LONG                          DPCAgentPID = 0;
/* Main Handler thread PID               */
LONG                          DPCFilePID = 0;
/* Packet Handler thread PID    */
LONG                          DPCModemPID = 0;
/* Modem Handler thread PID     */
LONG                          DPCAccessPID = 0;
/* Access thread PID                        */
LONG                          DPCInetPID = 0;
/* Turbo Internet thread PID             */
BYTE                          DPCName[] = {TxtMSG("\x03""DPC", 72)};
int                           ExitingFlag = FALSE;
/* All threads must exit                 */
LONG                          InetAsleep = FALSE;
/* Tinet needs to wake up flag  */
int                                   InReturnResources = FALSE;
/*
 * CRC table used by calccrc().
 */
static unsigned short crctab[256] = {
     0x0000, 0x1189, 0x2312, 0x329b, 0x4624, 0x57ad, 0x6536, 0x74bf,
     0x8c48, 0x9dc1, 0xaf5a, 0xbed3, 0xca6c, 0xdbe5, 0xe97e, 0xf8f7,
     0x1081, 0x0108, 0x3393, 0x221a, 0x56a5, 0x472c, 0x75b7, 0x643e,
```

```
        0x9cc9, 0x8d40, 0xbfdb, 0xae52, 0xdaed, 0xcb64, 0xf9ff, 0xe876,
        0x2102, 0x308b, 0x0210, 0x1399, 0x6726, 0x76af, 0x4434, 0x55bd,
        0xad4a, 0xbcc3, 0x8e58, 0x9fd1, 0xeb6e, 0xfae7, 0xc87c, 0xd9f5,
        0x3183, 0x200a, 0x1291, 0x0318, 0x77a7, 0x662e, 0x54b5, 0x453c,
        0xbdcb, 0xac42, 0x9ed9, 0x8f50, 0xfbef, 0xea66, 0xd8fd, 0xc974,
        0x4204, 0x538d, 0x6116, 0x709f, 0x0420, 0x15a9, 0x2732, 0x36bb,
        0xce4c, 0xdfc5, 0xed5e, 0xfcd7, 0x8868, 0x99e1, 0xab7a, 0xbaf3,
        0x5285, 0x430c, 0x7197, 0x601e, 0x14a1, 0x0528, 0x37b3, 0x263a,
        0xdecd, 0xcf44, 0xfddf, 0xec56, 0x98e9, 0x8960, 0xbbfb, 0xaa72,
        0x6306, 0x728f, 0x4014, 0x519d, 0x2522, 0x34ab, 0x0630, 0x17b9,
        0xef4e, 0xfec7, 0xcc5c, 0xddd5, 0xa96a, 0xb8e3, 0x8a78, 0x9bf1,
        0x7387, 0x620e, 0x5095, 0x411c, 0x35a3, 0x242a, 0x16b1, 0x0738,
        0xffcf, 0xee46, 0xdcdd, 0xcd54, 0xb9eb, 0xa862, 0x9af9, 0x8b70,
        0x8408, 0x9581, 0xa71a, 0xb693, 0xc22c, 0xd3a5, 0xe13e, 0xf0b7,
        0x0840, 0x19c9, 0x2b52, 0x3adb, 0x4e64, 0x5fed, 0x6d76, 0x7cff,
        0x9489, 0x8500, 0xb79b, 0xa612, 0xd2ad, 0xc324, 0xf1bf, 0xe036,
        0x18c1, 0x0948, 0x3bd3, 0x2a5a, 0x5ee5, 0x4f6c, 0x7df7, 0x6c7e,
        0xa50a, 0xb483, 0x8618, 0x9791, 0xe32e, 0xf2a7, 0xc03c, 0xd1b5,
        0x2942, 0x38cb, 0x0a50, 0x1bd9, 0x6f66, 0x7eef, 0x4c74, 0x5dfd,
        0xb58b, 0xa402, 0x9699, 0x8710, 0xf3af, 0xe226, 0xd0bd, 0xc134,
        0x39c3, 0x284a, 0x1ad1, 0x0b58, 0x7fe7, 0x6e6e, 0x5cf5, 0x4d7c,
        0xc60c, 0xd785, 0xe51e, 0xf497, 0x8028, 0x91a1, 0xa33a, 0xb2b3,
        0x4a44, 0x5bcd, 0x6956, 0x78df, 0x0c60, 0x1de9, 0x2f72, 0x3efb,
        0xd68d, 0xc704, 0xf59f, 0xe416, 0x90a9, 0x8120, 0xb3bb, 0xa232,
        0x5ac5, 0x4b4c, 0x79d7, 0x685e, 0x1ce1, 0x0d68, 0x3ff3, 0x2e7a,
        0xe70e, 0xf687, 0xc41c, 0xd595, 0xa12a, 0xb0a3, 0x8238, 0x93b1,
        0x6b46, 0x7acf, 0x4854, 0x59dd, 0x2d62, 0x3ceb, 0x0e70, 0x1ff9,
        0xf78f, 0xe606, 0xd49d, 0xc514, 0xb1ab, 0xa022, 0x92b9, 0x8330,
        0x7bc7, 0x6a4e, 0x58d5, 0x495c, 0x3de3, 0x2c6a, 0x1ef1, 0x0f78
          };


/* MLID Statistic globals */

int      GenericDescriptionTable[22+8] =
{
         InxMSG("Total packets sent:", 103),
         InxMSG("Total packets received:", 141),
         InxMSG("No ECB available count:", 148),
         InxMSG("Send packet too big count:", 156),
         InxMSG("Reserved:", 186),
         InxMSG("Receive packet overflow count:", 214),
         InxMSG("Receive packet too big count:", 215),
         InxMSG("Receive packet too small count:", 216),
         InxMSG("Send packet miscellaneous errors:", 217),
         InxMSG("Receive packet miscellaneous errors:", 218),
         InxMSG("Send packet retry count:", 223),
         InxMSG("Checksum errors:", 290),
         InxMSG("Hardware receive mismatch count:", 291),
         InxMSG("Total send OK byte count low:", 229),
         InxMSG("Total send OK byte count high:", 230),
         InxMSG("Total receive OK byte count low:", 231),
         InxMSG("Total receive OK byte count high:", 232),
         InxMSG("Total group address send count:", 233),
         InxMSG("Total group address receive count:", 251),
         InxMSG("Adapter reset count:", 252),
         InxMSG("Adapter operating time stamp:", 253),
         InxMSG("Adapter queue depth:", 255),
         InxMSG("Send OK single collision count:", 159),
         InxMSG("Send OK multiple collision count:", 256),
         InxMSG("Send OK but deferred", 264),
         InxMSG("Send abort from late collision", 265),
         InxMSG("Send abort from excess collision", 266),
         InxMSG("Send abort from carrier sense", 267),
         InxMSG("Send abort from excessive deferral", 268),
         InxMSG("Receive abort from bad frame alignment", 269)
```

```
);

#define STATS_DATA_WIDTH                13
#define FSD_DATA_WIDTH                  40
#define BORDER_WIDTH                            3
#define INDENT_WIDTH                            3


void ( *BackgroundFuncPtr )( LONG portalNumber ) = NULL;
LONG    BackgroundPortal;
int     GenericLineStart;
int GblDataCol;
struct DOSCountryInfoStruct     GblDOSCountryInfo;

int     DebugFlag       = FALSE;

/* The array nDeclination contains the Declination in degrees to add or
 * subtract from true azimuth to get magnetic azimuth.
 * The value 0 means that the location in not in mainland US
 */
static float nDeclination[]=
{
        0,  0,  4, 1,  0,  0,  0, -8,   0,   0,   0,   0,   0,   0,
        0,  0,  0, 1, -3, -4, -6, -8, -10, -11,  12, -13,   0,   0,
        0,  0,  5, 2, -2, -5, -6, -9, -11, -12, -13, -13, -13, -14,
        0,  7,  5, 2, -1, -4, -6, -8, -12, -12, -14, -15, -16, -16,
        0, 12,  9, 6, -1, -4, -6, -9, -11, -12, -14, -16, -17, -17,
       18, 15,  0, 7,  2, -3, -6, -9, -12, -13, -15, -18, -19, -19,
        0,  0,  0, 0,  0, -1, -6, -9, -13, -15, -18, -20, -21, -22
};

#ifdef LOG_ECB_ACTIVITY
int DPC_TGID;
LogHandle LogClientHandle;
EventHandle LogECBHandle;
#endif /* LOG_ECB_ACTIVITY */

/*

 * Local function prototypes.
 */
void ReturnResources(int sig);
/************************************************************************
 *
 *      ExitHandler(void *handle)
 *
 *      Description:
 *              This routine is called when the user hits Alt-F10 to exit
 *              the utility. We will attempt to verify with the user that
 *              they really want to exit.
 *
 *      Input:
 *              handle          - NUT handle
 *
 *      Output:
 *              nothing
 *
 *      Returns:
 *              nothing
 *
 ************************************************************************/

void    ExitHandler(void        *handle)
{
        if (NWSConfirm(InxMSG("Exit DSDEBUG?", 88), 0, 0, TRUE, NULL,
                handle, NULL) == TRUE)
```

```
        {
                ReturnResources(1);
        exit(1);
        }
}

static void NoSortHandler(LIST *head, LIST *tail, NUTInfo *handle)
{
        head = head;
        tail = tail;
        handle = handle;

        return;
}

/************************************************************************
 *
 *      calccrc(WORD crc,
 *                      BYTE *cp,
 *                      LONG len)
 *
 *      Description:
 *              This routine calculates a CRC value for the text passed
 *              in. It uses a CRC substitution table for effeciency.
 *              Its used by the Slip transmit routine.
 *
 *      Input:
 *              crc                                     - Initial value of the C
RC.
 *              cp                                              - string to calc
ulate CRC against
 *              len                                     - length of the string
 *
 *      Output:
 *              nothing
 *
 *      Returns:
 *              16-bit CRC value
 *
 ************************************************************************/

WORD calccrc(WORD crc, BYTE *cp, LONG len)
{
        while(len--)
                crc = (crc >> 8) ^ crctab[(crc ^ *cp++) & 0xff];

        return(crc);
}

/************************************************************************
 *
 *      SlipSend(char *pdata,
 *                      LONG sz,
 *                      int cas,
 *                      int timeout)
 *
 *      Description:
 *              This routine builds a SLIP envelope around the message
 *              passed in, escapes any HDLC characters in the message,
 *              and sends it to the modem.
 *
 *      Input:
 *              pdata                                   - Pointer to the message
 to send
 *              sz                                              - length of the
message passed in
```

```
*               cas                                     - 1 if send to cas, 0 if
 send to hub
*               timeout                         - inactivity timeout in seconds
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
***************************************************************************/

void SlipSend( char *pdata, LONG sz, int cas, int timeout)
{
        LroEsPkt_t txcas;
        LroAsPkt_t txhub;
        LONG tmplen;
        WORD crc;
        BYTE *pi, *po;
        BYTE slip_data[3072];
        int is, os;

        if (cas)
        {
                /* initialize cas transfer */
                CMovB(pdata, txcas.data, sz);
                txcas.length = sz + (txcas.data - ((BYTE *)&txcas));
                tmplen = txcas.length;
                txcas.length |= 0x8000;
                crc = calccrc(INITCRC, (BYTE *)&txcas, tmplen);
                txcas.data[sz] = crc & 0xff;
                txcas.data[sz+1] = (crc >> 8) & 0xff;
                pi = (BYTE *) &txcas;
        }
        else
        {
                /* initialize hub transfer */
                CMovB(pdata, txhub.data, sz);
                txhub.length = sz + (txhub.data - ((BYTE *)&txhub));
                txhub.filler1 = txhub.channel = 0;
                txhub.control = 0x8000;
                tmplen = txhub.length;
                crc = calccrc(INITCRC, (BYTE *)&txhub, tmplen);
                txhub.data[sz] = crc & 0xFF;
                txhub.data[sz + 1] = (crc >> 8) & 0xff;
                pi = (BYTE *) &txhub;
        }

        /* Start out with SLIP start character */
        po = slip_data;
        *po++ = END;

        /* Escape any special SLIP characters that may be in the message */
        for (is = 0, os = 1; is < (tmplen + 2); is++, os++, pi++, po++)
        {
                switch(*pi)
                {
                        case END:
                                *po++ = ESC;
                                *po = ESC_END;
                                os++;
                                break;
                        case ESC:
                                *po++ = ESC;
                                *po = ESC_ESC;
                                os++;
```

```
                                break;
                        default:
                                *po = *pi;
                                break;
                }
        }

        /* Finish packet off with SLIP END character */
        *po = END;
        os++;

        /* Send message to the modem thread */
        DloSend(slip_data, os, timeout);
}

/***************************************************************************
*
*       EXPORTED FUNCTION
*
*       DPCGetBoard(LONG *board,
*                       LONG *controlEntry)
*
*       Description:
*               This routine returns the DPC MLID logical board number
*               and control entry address to be used to send IOCTL
*               requests to the DPC mlid. The IoctlMlid pragma can
*               then be used to make the request.
*
*       Input:
*               board                                   - Pointer to where to st
ore board number
*               controlEntry            - Pointer to where to store control entr
y address
*
*       Output:
*               board and controlEntry filled in if successful
*
*       Returns:
*               0               if MLID is active
*
***************************************************************************/

LONG DPCGetBoard(LONG *board, LONG *controlEntry)
{
        if (DIOBoard == 0)
                return(-1);

        *board = DIOBoard;
        *controlEntry = DIOControlEntry;
        return(0);
}


void
CreateStringWithCommas( LONG number, BYTE *buffer, char *format )
{
        int     i;
        int     j;
        int     found;
        int     length;
        int     commasAdded = 0;
        BYTE    tmpBuf[ 128 ];
        BYTE    *tmpPtr;

        NWsprintf( tmpBuf, format, number);
        for ( i = ( length = CStrLen( tmpBuf ) ), found = 0; i >= 0; i-- )
        {
                if ( ( tmpBuf[ i ] >= '0' ) && ( tmpBuf[ i ] <= '9' ) )
                {
                        /* we have a digit */

                        if ( ++found > 3 )
                        {
                                found = 1;

                                /* shift the string one to the right */

                                for ( j = ++length; j > i; j-- )
                                        tmpBuf[ j ] = tmpBuf[ j - 1 ];

                                tmpBuf[ i + 1 ] = GblDOSCountryInfo.thousandSep[
 0 ];

                                commasAdded++;
                        }
                }
        }

        /*
        ** Adjust the length of the string back to its original if there are
        ** leading spaces.
        */

        for ( i = 0, tmpPtr = tmpBuf; i < commasAdded; i++ )
        {
                if ( *tmpPtr == ' ' )
                        tmpPtr++;
        }
        CStrCpy( buffer, tmpPtr );
}


void
SecondsToDateAndTime( LONG seconds, BYTE *buff)
{
        char *ascBuf;

        ascBuf = asctime(localtime((time_t *)&seconds));
        CMovB(ascBuf, buff, 26);
        buff[24] = 0;
}

void
FormatElapsedTime( LONG seconds, LONG tenths, BYTE *buff )
{
        LONG    minutes, hours, days;

        /* convert secs to minutes */

        minutes = seconds / 60;
        seconds = seconds % 60;
        hours = minutes / 60;
        minutes = minutes % 60;
        days = hours / 24;
        hours = hours % 24;
        if ( days > 0 )
        {
                NWsprintf
                (
                        buff,
                        MSG("%d %c%02d%c%02d%c%02d%c%01d", 293),
                        days,
                        GblDOSCountryInfo.timeSep[ 0 ],
```

```
                        hours,
                        GblDOSCountryInfo.timeSep[ 0 ],
                        minutes,
                        GblDOSCountryInfo.timeSep[ 0 ],
                        seconds,
                        GblDOSCountryInfo.decimalSep[ 0 ],
                        tenths
                );
        }
        else if ( hours > 0 )
        {
                NWsprintf
                (
                        buff,
                        MSG("%2d%c%02d%c%02d%c%01d", 294),
                        hours,
                        GblDOSCountryInfo.timeSep[ 0 ],
                        minutes,
                        GblDOSCountryInfo.timeSep[ 0 ],
                        seconds,
                        GblDOSCountryInfo.decimalSep[ 0 ],
                        tenths
                );
        }
        else if ( minutes > 0 )
        {
                NWsprintf
                (
                        buff,
                        MSG("%2d%c%02d%c%01d", 295),
                        minutes,
                        GblDOSCountryInfo.timeSep[ 0 ],
                        seconds,
                        GblDOSCountryInfo.decimalSep[ 0 ],
                        tenths
                );
        }
        else
        {
                NWsprintf
                (
                        buff,
                        MSG("%2d%c%01d", 296),
                        seconds,
                        GblDOSCountryInfo.decimalSep[ 0 ],
                        tenths
                );
        }
}

void HandleScrollablePortal(PCB *portal)
{
        int escapeFlag = 0;
        LONG    keyType;
        BYTE    ch;
        int             updateDisplay = TRUE;
        LONG    virtualHeight;
        LONG    portalHeight;
        LONG    bottomLine;
        LONG    curPos;
        LONG    vLine;

        /*
        ** The values for portal->portalHeight and portal->virtualHeight are the
        ** only two that we deal with that are 1-based.  All the rest are 0-base
d,
        ** so we will use local copies of these variables adjusted to fit in wit
h
        ** the rest in the PCB structure.
        **/

        virtualHeight = portal->virtualHeight - 1;
        portalHeight = portal->portalHeight - 1;

        /*
        ** Other initializations.
        */

        bottomLine = virtualHeight - portalHeight;


        while(!escapeFlag)
        {

                if ( updateDisplay == TRUE )
                {
                        /*
                        ** The last iteration of the loop made a change, so redr
aw the
                        ** portal.
                        */

                        if ( portal->verticalScroll == SCROLL_ON )
                        {
                                /*
                                ** Adjust the vertical thumb on the right.border
.
                                */

                                if ( portal->cursorLine == 0 )
                                {
                                        if ( virtualHeight <= portalHeight )
                                                curPos = 100;
                                        else
                                                curPos = 0;
                                }
                                else
                                {
                                        if ( portal->cursorLine >= bottomLine )
                                                curPos = 100;
                                        else
                                        {
                                                vLine = bottomLine;
                                                if ( vLine == 0 )
                                                        vLine = 1;
/* avoid divide by 0 */

                                                curPos = ( portal->cursorLine *
100 ) / vLine;
                                        }
                                }

                                /*
                                ** Erase the vertical thumb at its last position
.
                                */

                                portal->showScrollBars &= ~VERTICAL_SCROLL_MASK;

                                /*
                                ** Display the vertical thumb at its new positio
n.
```

```
                                */

                                portal->showScrollBars |= curPos << VERTICAL_SCR
OLL_SHIFT;
                        }

                        NWSUpdatePortal( portal );
                        updateDisplay = FALSE;
                }

                NWSGetKey( &keyType, &ch, NUTHandle );
                switch ( keyType )
                {
                        case K_UP:

                                /*
                                ** Scroll the portal up one line.
                                */

                                if ( portal->virtualLine > 0 )
                                {
                                        /*
                                        ** We're not at the top, so we can scrol
l up one line.
                                        */

                                        portal->virtualLine--;
                                        portal->cursorLine = portal->virtualLine
;
                                        updateDisplay = TRUE;
                                }
        break;

                        case K_DOWN:

                                /*
                                ** Scroll the portal down one line.
                                */

                                if ( portal->virtualLine < bottomLine )
                                {
                                        /*
                                        ** We're not at the bottom, so we can sc
roll down one line.
                                        */

                                        portal->virtualLine++;
                                        portal->cursorLine = portal->virtualLine
;
                                        updateDisplay = TRUE;
                                }
        break;

                        case K_PUP:

                                /*
                                ** Move the portal up one page.
                                */

                                if ( portal->cursorLine > 0 )
                                {
                                        LONG    delta;

                                        /*
                                        ** We're not at the top, so we can move
up.  However, we need
```

```
                                        ** to figure out how much we can move up

                                        */

                                        if ( portal->cursorLine > portalHeight )
                                                delta = portalHeight;
                                        else
                                                delta = portal->cursorLine;

                                        portal->cursorLine -= delta;
                                        if ( portal->cursorLine < portal->virtua
lLine )
                                                portal->virtualLine = portal->cu
rsorLine;

                                        updateDisplay = TRUE;
                                }
        break;

                        case K_PDOWN:

                                /*
                                ** Move the portal down one page.
                                */

                                if ( portal->cursorLine < virtualHeight )
                                {
                                        LONG    delta;
                                        LONG    newCurrentLine;

                                        /*
                                        ** We're not at the bottom, so we can mo
ve down.  However,
                                        ** we need to figure out how much we can
 move down.
                                        */

                                        delta = virtualHeight - portal->cursorLi
ne;

                                        if ( delta > portalHeight )
                                                delta = portalHeight;

                                        newCurrentLine = portal->cursorLine + de
lta;

                                        delta = virtualHeight - portalHeight;
                                        if ( newCurrentLine > delta )
                                                portal->virtualLine = delta;
                                        else
                                                portal->virtualLine = newCurrent
Line;

                                        portal->cursorLine = portal->virtualLine
;

                                        updateDisplay = TRUE;
                                }
                                break;

                        case K_SUP:

                                /*
                                ** <Ctrl-PgUp> takes us to the top of the portal
.
                                */

                                portal->virtualLine = 0;
                                portal->cursorLine = 0;
                                updateDisplay = TRUE;
```

```
                                break;

                        case K_SDOWN:

                                /*
                                ** <Ctrl-PgDn> takes us to the bottom of the por
tal.
                                */

                                portal->cursorLine = virtualHeight;
                                portal->virtualLine = portal->cursorLine - porta
lHeight;

                                updateDisplay = TRUE;
                                break;


                        case K_ESCAPE:
                                        escapeFlag = 1;
                                break;

                }
        }

}

LONG    DPCGetSignalStrength(void)
{
        struct DriverStatsStructure *stats;
        CustVars                *customPtr;
        LONG signal;
        int beamSize, beamPercent;

        if (DPCGetMLIDStats(&stats))
                return(0);

        customPtr = (CustVars *)(&stats->CustomVariableCount);
        signal = customPtr->CustomVariable[0];

        if (signal >= 200)
        signal = (2 * (signal - 200)) + 60;
        else
    signal = 0;

        beamSize  = (int)((signal - 60L)/2L);
        beamSize *= 5;
        beamSize /= 4;

        if(beamSize < 0)
        {
                beamSize = 0;
        }
        else if(beamSize >= MAX_BEAM)
        {
                beamSize = MAX_BEAM;
        }

        beamPercent = (100*beamSize) / MAX_BEAM;

        return(beamPercent);
}

void UpdateStatsInformation(LONG portal)
{
        PCB             *portalPtr;
        struct DriverStatsStructure *stats;
        int     line, count;
        LONG                    *statsPtr;
        int numGenerics;
        BYTE                    string[80];
        LONG                    mask;
        LONG                    seconds;
        LONG                    tenths;
        CustVars                *customPtr;
        BYTE                    *ptr;

        NWSGetPCB( &portalPtr, portal, NUTHandle );

        if (DPCGetMLIDStats(&stats))
        {
                AddKey( NUTHandle->screenID, ENTER_KEY, 0, 0, 0 );
                return;
        }


        /* do generic stuff */

        line = GenericLineStart;
        statsPtr = &( stats->NotSupportedMask );
        numGenerics = stats->GenericVariableCount;
        mask = *statsPtr++;

        for ( count = 0; count < numGenerics; count++ )
        {
                if ( mask & ( 0x80000000 >> ( count & 0x1f ) ) )
                {
                        /* not supported */
                        NWsprintf( string, MSG("%13.13s", 301), MSG("Not support
ed", 298) );
                        NWSShowPortalLine
                        (
                                line++,
                                GblDataCol,
                                string,
                                STATS_DATA_WIDTH,
                                portalPtr
                        );
                        statsPtr++;
                }
                else
                {
                        if ( count == 20 )
                        {
                                BYTE    tmpString[ 80 ];

                                /*
                                ** This is the operating time stamp, which needs
 a different
                                ** output format.
                                */

                                ConvertTicksToSeconds( *statsPtr++, &seconds, &t
enths );

                                FormatElapsedTime( seconds, tenths, tmpString );
                                NWsprintf( string, MSG("%13.13s", 302), tmpStrin
g );
                        }
                        else
                        {
                                CreateStringWithCommas
                                (
                                        *statsPtr++,
```

```
                                        string,
                                        MSG("%13lu", 303)
                                );
                        }
                        NWSShowPortalLine
                        (
                                line++,
                                GblDataCol,
                                string,
                                STATS_DATA_WIDTH,
                                portalPtr
                        );
                }
        }

        /* do custom stuff */

        line += 2;
        customPtr = (CustVars *)(&stats->CustomVariableCount);
        ptr = (BYTE *)(customPtr->CustomVariable[customPtr->CustomVariableCount]
);
        ptr += sizeof(WORD);

        for ( count = 0; count < customPtr->CustomVariableCount; count++ )
        {
                CreateStringWithCommas
                (
                        customPtr->CustomVariable[ count ],
                        string,
                        MSG("%13lu", 299)
                );
                NWSShowPortalLine( line++, GblDataCol, string, STATS_DATA_WIDTH,
 portalPtr );
        }

        NWSUpdatePortal( portalPtr );

}


void SignalMeter(void) {
  int   exitNow = FALSE;
  LONG type;
  BYTE value;
  int signal = 0;
  int   oldSignal = 0;
  int   avgSqf = 0;
  int beamSize;
  int beamPercent;
  int   bLock = FALSE;
  int rxFreq;
  struct DriverStatsStructure *stats;
  CustVars                *customPtr;
  char  freqStr[80];
  char  aveStr[80];
  char  signalStr[80];
  int sound_gap = 0;

  while(!exitNow) {
    if (DPCGetMLIDStats(&stats)) {
      type = signal = 0;
      rxFreq = 0;
    }
    else {
      customPtr = (CustVars *)(&stats->CustomVariableCount);
      type = signal = customPtr->CustomVariable[0];
    rxFreq = customPtr->CustomVariable[1];
  }

  NWsprintf(freqStr, MSG("     Frequency : %d", 345), rxFreq / 10);

  if (signal >= 200)
    signal = (2 * (signal - 200)) + 60;
  else
    signal = 0;

  if (avgSqf == 0L)
    avgSqf = 100L * signal;
  avgSqf = ((avgSqf * 19L) + (100L * (unsigned long) signal)) / 20L;

  beamSize  = (int)((avgSqf/100L - 60L)/2L);
  beamSize *= 5;
  beamSize /= 4;

  if(beamSize < 0)
    beamSize = 0;
  else if(beamSize >= MAX_BEAM)
    beamSize = MAX_BEAM - 1;

  beamPercent = (100*beamSize) / MAX_BEAM;

  if(oldSignal != signal) {
    if(signal < MIN_SQF_VAL)
      bLock = FALSE;
    else
      bLock = TRUE;
    oldSignal = signal;
  }

  NWsprintf(aveStr, MSG("   Average SQF : %d", 346),
           signal ? avgSqf / 100L : 0);

  NWsprintf(signalStr, MSG("Signal Quality : %d (%d%%)            %s", 347),
           signal,
           beamPercent,
           bLock ? MSG("(Signal Locked)", 348) : MSG("(Signal Not Locked)", 3
49));

  DisplayThrottle(MSG("Satellite Dish Signal Strength Meter", 341),
                  beamSize,
                  MAX_BEAM,
                  freqStr,
                  aveStr,
                  signalStr);

  if (--sound_gap <= 0) {
    /* calculate frequency based on raw signal strength */
    signal = 200 + type;
    /* adjust for 8253-5 timer chip output */
    signal = 1193180 / signal;

    /* turn on sound */
    outp(67, 182);
    outp(66, signal % 256);
    outp(66, signal / 256);
    outp(97, inp(97) | 0x03);

    delay(300);

    /* turn off sound */
    outp(97, inp(97) & ~0x03);
```

```
      sound_gap = (110 - beamPercent) / 17;
    }

    delay(150);

    if (NWSKeyStatus(NUTHandle)) {
      NWSGetKey(&type, &value, NUTHandle);
      if ((type == K_ESCAPE) || (type == K_AF10))
        {
          exitNow = TRUE;
        }
    }
  }

  /* Destroy the throttle portal */
  DisplayThrottle(MSG("Satellite Dish Signal Strength Meter", 351),
                  MAX_BEAM,
                  MAX_BEAM,
                  freqStr,
                  aveStr,
                  signalStr);

}

void GetRegionName(char *regionName)
{
        FILE *fp;
        char szTmp[128];
        char *src, *dest;
        LONG len;

        fp = fopen(MSG("country.ini", 635), MSG("r", 636));
        len = strlen(MSG("RegionName=", 637));
        *regionName = 0;
        if (fp != NULL)
        {
                while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
                {
                        if ( (strnicmp(szTmp, MSG("RegionName=", 638), len)) ==
0)
                        {
                                fclose(fp);
                                src = &szTmp[len];
                                dest = regionName;
                                while(*src != 0 && *src != ' ' && *src != '\r' &
& *src != '\n')
                                {
                                        *dest = *src;
                                        src++;
                                        dest++;
                                }
                                *dest = 0;
                                return;
                        }
                }
                fclose(fp);
        }

}

void GetCountry(char *countryName)
{
        FILE *fp;
        char szTmp[128];
        char *src, *dest;
        LONG len;
        fp = fopen(MSG("country.ini", 639), MSG("r", 640));
        len = strlen(MSG("Name=", 641));
        *countryName = 0;
        if (fp != NULL)
        {
                while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
                {
                        if ( (strnicmp(szTmp, MSG("Name=", 642), len)) == 0)
                        {
                                fclose(fp);
                                src = &szTmp[len];
                                dest = countryName;
                                while(*src != 0 && *src != ' ' && *src != '\r' &
& *src != '\n')
                                {
                                        *dest = *src;
                                        src++;
                                        dest++;
                                }
                                *dest = 0;
                                return;
                        }
                }
                fclose(fp);
        }

}

void GetDefaultService(char *serviceName)
{
        FILE *fp;
        char szTmp[128];
        char *src, *dest;
        LONG len;

        fp = fopen(MSG("country.ini", 626), MSG("r", 627));
        len = strlen(MSG("Service1=", 628));
        *serviceName = 0;
        if (fp != NULL)
        {
                while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
                {
                        if ( (strnicmp(szTmp, MSG("Service1=", 185), len)) == 0)
                        {
                                fclose(fp);
                                src = &szTmp[len];
                                dest = serviceName;
                                while(*src != 0 && *src != ' ' && *src != '\r' &
& *src != '\n')
                                {
                                        *dest = *src;
                                        src++;
                                        dest++;
                                }
                                *dest = 0;
                                return;
                        }
                }
                fclose(fp);
        }

}

typedef struct
{
```

```
        char    name[20];
        LONG    longitude;
        LONG    eastFlag;                       /* 1 for east, 0 for west */
        LONG    frequency;
        LONG    horzFlag;           /* 1 for horz, 0 for vert polarization */
        LONG    cityLongDegrees;
        LONG    cityLongMinutes;
        LONG    cityLatDegrees;
        LONG    cityLatMinutes;
        LONG    cityEastFlag;
        LONG    cityNorthFlag;
} SatelliteInfo;

typedef struct
{
        float   elevation;
        float   trueAzimuth;
        float   magAzimuth;
        float   polarization;
} DishInfo;


void ParseSatelliteInfo(char *satName, int nameContainsInfo, SatelliteInfo *sat)
{
        FILE *fp;
        char szTmp[128];
        char *fptr;
        char *name, *cptr;
        char *ascPtr;
        char ascBuf[10];

        if (nameContainsInfo == FALSE)
        {
                fp = fopen(MSG("country.ini", 644), MSG("r", 629));
                if (fp != NULL)
                {
                        while ((fptr = fgets(szTmp, sizeof(szTmp) - 1, fp)) != N
ULL)
                        {
                                if ( (strnicmp(szTmp, satName, strlen(satName)))
 == 0)
                                        break;
                        }
                }
                if (fp)
                        fclose(fp);
                if (!fptr)
                        return;
                cptr = szTmp;
        }
        else
                cptr = satName;

        name = sat->name;
        while(*cptr != '=')
                *name++ = *cptr++;
        *name = 0;

        cptr++;
        while(*cptr == ' ')
                cptr++;

        ascPtr = ascBuf;
        while(*cptr != ',')
                *ascPtr++ = *cptr++;
        cptr++;
        *ascPtr = 0;
        sat->longitude = atoi(ascBuf);

        if (*cptr == 'e' || *cptr == 'E')
                sat->eastFlag = 1;
        else
                sat->eastFlag = 0;

        while(*cptr != ',')
                cptr++;
        cptr++;

        ascPtr = ascBuf;
        while(*cptr != ',')
                *ascPtr++ = *cptr++;
        cptr++;
        *ascPtr = 0;
        sat->frequency = atoi(ascBuf);

        if (*cptr == 'h' || *cptr == 'H')
                sat->horzFlag = 1;
        else
                sat->horzFlag = 0;

}

void GetDefaultSatellite(SatelliteInfo *sat)
{
        FILE *fp;
        char szTmp[128];
        LONG len;
        char *ccode;

        fp = fopen(MSG("country.ini", 632), MSG("r", 633));
        len = strlen(MSG("[Hughes Networks]", 630));
        if (fp != NULL)
        {
                while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
                {
                        if ( (strnicmp(szTmp, MSG("[Hughes Networks]", 631), len
)) == 0)
                        {
                                ccode = fgets(szTmp, sizeof(szTmp) - 1, fp);
                                fclose(fp);
                                if (ccode == NULL)
                                        return;

                                ParseSatelliteInfo(szTmp, TRUE, sat);
                                return;
                        }
                }
                fclose(fp);
        }

}

int NewCalculationFlag = 0;

LONG    ChangeSatellite(FIELD *fieldPtr, int key, int *changed, NUTInfo *handle)
{
        FILE *fp;
        SatelliteInfo *sat;
        LIST    *listPtr = NULL;
        LONG ccode;
        LONG rcode = K_SELECT;
```

```
        char *fstr;
        char szTmp[128];
        char *szPtr;
        int rows = 0, cols = 0, len;
        void (*oldSortFunction)(LIST *, LIST *, NUTInfo *);

        key = key;
        changed = changed;
        handle = handle;

        sat = (SatelliteInfo *)fieldPtr->customData;

        NWSGetListSortFunction(NUTHandle, &oldSortFunction);
        NWSSetListSortFunction(NUTHandle, NoSortHandler);

        if (NWSPushList(NUTHandle) == 0)
        {
                NWSSetListSortFunction(NUTHandle, oldSortFunction);
                return rcode;
        }

        NWSInitList(NUTHandle, NULL);

        fp = fopen(MSG("country.ini", 634), MSG("r", 667));
        len = strlen(MSG("[Hughes Networks]", 668));
        if (fp != NULL)
        {
                while ((fstr = fgets(szTmp, sizeof(szTmp) - 1, fp)) != NULL)
                {
                        if ( (strnicmp(szTmp, MSG("[Hughes Networks]", 205), len
)) == 0)
                                break;
                }
        }
        if (fstr != NULL)
        {
                while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
                {
                        if (*szTmp == ' ' || *szTmp == '\r' || *szTmp == '\n')
                                break;
                        szPtr = szTmp;
                        while (*szPtr != '=')
                                szPtr++;
                        *szPtr = 0;
                        AppendToList(szTmp, 0, &rows, &cols);
                }
        }

        if (rows == 0)
                goto ChangeSatExit;

        ccode = NWSList(
                        InxMSG("Choose Satellite", 648),
                        12, 40,
                        (rows > 16) ? 16 : rows,
/* Height */
                        (cols < 20) ? 20 : cols,
/* Width */
                        M_ESCAPE | M_SELECT,
                        &listPtr,
                        NUTHandle, NULL,
                        NULL, NULL);

        if (ccode == M_SELECT)
        {
                ParseSatelliteInfo(listPtr->text, FALSE, sat);
                NewCalculationFlag = 1;
                rcode = K_ESCAPE;
        }

ChangeSatExit:
        NWSDestroyList(NUTHandle);
        NWSPopList(NUTHandle);
        NWSSetListSortFunction(NUTHandle, oldSortFunction);
        return rcode;
}

LONG    ComputeHotSpot(FIELD *fp, int key, int *changed, NUTInfo *handle)
{
        fp = fp;
        key = key;
        changed = changed;
        handle = handle;

        NewCalculationFlag = 1;
        return K_ESCAPE;
}

int     LongitudeHemisphereHandler(int option, void *parameter)
{
        parameter = parameter;
        return option;
}

int     PolarizationHandler(int option, void *parameter)
{
        parameter = parameter;
        return option;
}

int     GroundLatHemHandler(int option, void *parameter)
{
        parameter = parameter;
        return option;
}

int     GroundLongHemHandler(int option, void *parameter)
{
        parameter = parameter;
        return option;
}

float SGN(float num)
{
        if(num < 0)
                return -1;
        return 1;
}

void CalculateDish(SatelliteInfo *sat, DishInfo *dish)
{

        float LW,LN,ZR,YR,XR,DIST,EL,PO,AZ,TRAZ,S,A;
        float SD,RD,RM,LD,LM;
static const float M_PI = 3.14159;
static const float RAD = 6378.388;
static const float ALT = 42164.24;
        int count = 4;

//      SD = p->dSatLong;
//      RD = p->dRemLongDeg;
//      RM = p->dRemLongMin;
```

```
//      LD = p->dRemLatDeg;
//      LM = p->dRemLatMin;

        SD = (float)sat->longitude;
        RD = (float)sat->cityLongDegrees;
        RM = (float)sat->cityLongMinutes;
        LD = (float)sat->cityLatDegrees;
        LM = (float)sat->cityLatMinutes;

        SD = fabs(SD);
//      if (p->cSatLong == 'E')
        if (sat->eastFlag)
                SD = -SD;

        RD = fabs(RD);
        RM = fabs(RM);
//      if (p->cRemLong == 'E')
        if (sat->cityEastFlag)
        {
                RD = -RD;
                RM = -RM;
        }

        LD = fabs(LD);
        LM = fabs(LM);

//      if (p->cRemHem == 'S')
        if (sat->cityNorthFlag == 0)
        {
                LD = -LD;
                LM = -LM;
        }
        LW = ((RD + RM / 60) - SD) * M_PI / 180;

        if (LW == 0)
                LW = .00001;
        LN = (LD + LM / 60) * M_PI / 180;

        if (LN == 0)
                LN = .00001;

        ZR = RAD * sin(LN);
        YR = RAD * cos(LN) * cos(LW);
        XR = RAD * cos(LN) * sin(LW);
        DIST = sqrt(XR * XR + (ALT - YR) * (ALT - YR) + ZR * ZR);

        EL = (ALT * YR - RAD * RAD) / (RAD * DIST);
        EL = atan(EL / sqrt(1 - EL * EL)) * 180 / M_PI;

        EL = (10 * EL + .5 * SGN(EL)) / 10;

        A = 0;

        if (LN * LW > 0)
                A = 2 * M_PI;
        if (LN > 0)
                A = M_PI;

        AZ = (A - atan(tan(LW) / sin(LN))) * 180 / M_PI;
        AZ = floor(10 * AZ + .5 * SGN(AZ)) / 10;
        TRAZ = AZ;

        /* Executed for US Mainland only */
        /* Declination correction to TRUE Azimuth */
        if(((LD > 23) && (LD < 50)) && ((RD > 70) && (RD < 125)))
        {
                LD = floor(LD/4) - 6;
                RD = ceil(RD/4) - 18;
                if(!nDeclination[(int)(LD*14+RD)])
                        count--;
                if(!nDeclination[(int)(LD*14+RD-1)])
                        count--;
                if(!nDeclination[(int)((LD+1)*14+RD-1)])
                        count--;
                if(!nDeclination[(int)((LD+1)*14+RD)])
                        count--;
                if(count)
                {
                        AZ = AZ+(nDeclination[(int)(LD*14+RD)]
                                +nDeclination[(int)(LD*14+RD-1)]
                                +nDeclination[(int)((LD+1)*14+RD)]
                                +nDeclination[(int)((LD+1)*14+RD-1)])/count;
                }
        }

        if (AZ >= 360) AZ = AZ - 360;

        if (AZ < 0 ) AZ = AZ + 360;


        S = tan(LN) / sin(LW);

        PO = atan(S);
        /* printf("S=%lf LN=%lf LW=%lf PO=%lf\n",S,LN,LW,PO); */

        PO = fabs(PO);
        PO = M_PI / 2.0 - PO;
        PO = PO * SGN(S) * 180 / M_PI;
        PO = floor(10 * PO + .5 * SGN(PO)) / 10;

//      p->dRemElev = EL;
//      p->dRemMagAz = AZ;
//      p->dRemTrueAz = TRAZ;
//      p->dRemPolar = -PO;
        dish->elevation = EL;
        dish->magAzimuth = AZ;
        dish->trueAzimuth = TRAZ;
        dish->polarization = -PO;

}

void ComputeCity(char *region, char *city, LONG latLong)
{
        FIELD *fp;
        char country[20], countryStr[30];
        char regionStr[30], cityStr[30];
        char service[30], serviceStr[30];
        char satelliteStr[50];
        char elevationStr[50], trueAzimuthStr[50];
        char magAzimuthStr[50], polarizationStr[50];
        SatelliteInfo sat;
        DishInfo dish;
        int i;
        int start;
        MFCONTROL *mfctl0, *mfctl1, *mfctl2, *mfctl3;

calculateAgain:
        NWSInitForm(NUTHandle);

        if (NewCalculationFlag == 0)
        {
```

```
                sat.cityLatDegrees = (latLong >> 24) & 0xff;
                sat.cityLatMinutes = (latLong >> 16) & 0xff;
                sat.cityLongDegrees = (latLong >> 8) & 0xff;
                sat.cityLongMinutes = latLong & 0xff;
                GetDefaultSatellite(&sat);
                sat.cityEastFlag = sat.eastFlag;
                sat.cityNorthFlag = 1;
        }
        NewCalculationFlag = 0;

        i = 0;
        GetCountry(country);
        NWsprintf(countryStr, MSG("Country : %s", 488), country);
        NWSAppendCommentField(i, 2, countryStr, NUTHandle);

        NWsprintf(regionStr, MSG("Region : %s", 712), region);
        NWSAppendCommentField(i, 36, regionStr, NUTHandle);

        i++;
        NWsprintf(cityStr, MSG("City    : %s", 713), city);
        NWSAppendCommentField(i, 2, cityStr, NUTHandle);

        GetDefaultService(service);
        NWsprintf(serviceStr, MSG("Service : %s", 714), service);
        NWSAppendCommentField(i, 36, serviceStr, NUTHandle);

        i++;
        NWsprintf(satelliteStr, MSG("Satellite : %s", 649), sat.name);
        start = (78 - strlen(satelliteStr)) / 2;
        fp = NWSAppendHotSpotField(i, start, NORMAL_FIELD, satelliteStr,
                                        ChangeSatellite, NUTHandle);
        fp->customData = &sat;

        i++;
        NWSAppendCommentField(i, 2, MSG("Satellite Longitude    : ", 715), NUTHa
ndle);
        NWSAppendIntegerField(i, 27, NORMAL_FIELD, (int *)&sat.longitude, 1, 180
, F_NO_HELP, NUTHandle);

        NWSAppendCommentField(i, 34, MSG("Hemisphere : ", 716), NUTHandle);
        mfctl0 = NWSInitMenuField(InxMSG("Satellite Longitude Hemisphere", 717),
 10, 40, LongitudeHemisphereHandler, NUTHandle);
        NWSAppendToMenuField(mfctl0, InxMSG("West", 718), 0, NUTHandle);
        NWSAppendToMenuField(mfctl0, InxMSG("East", 719), 1, NUTHandle);
        NWSAppendMenuField(i, 47, NORMAL_FIELD, (int *)&sat.eastFlag, mfctl0, NU
LL, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Satellite Polarization : ", 489), NUTHa
ndle);
        mfctl1 = NWSInitMenuField(InxMSG("Satellite Polarization", 490), 10, 40,
 PolarizationHandler, NUTHandle);
        NWSAppendToMenuField(mfctl1, InxMSG("Vert", 531), 0, NUTHandle);
        NWSAppendToMenuField(mfctl1, InxMSG("Horz", 539), 1, NUTHandle);
        NWSAppendMenuField(i, 27, NORMAL_FIELD, (int *)&sat.horzFlag, mfctl1, NU
LL, NUTHandle);

        NWSAppendCommentField(i, 34, MSG("Frequency  : ", 541), NUTHandle);
        NWSAppendIntegerField(i, 47, NORMAL_FIELD, (int *)&sat.frequency, 1, 100
00, F_NO_HELP, NUTHandle);

        i+=2;
        NWSAppendCommentField(i, 2, MSG("Ground Longitude degrees : ", 543), NUT
Handle);
        NWSAppendIntegerField(i, 30, NORMAL_FIELD, (int *)&sat.cityLongDegrees,
1, 180, F_NO_HELP, NUTHandle);
```

```
        NWSAppendCommentField(i, 40, MSG("minutes : ", 545), NUTHandle);
        NWSAppendIntegerField(i, 50, NORMAL_FIELD, (int *)&sat.cityLongMinutes,
1, 180, F_NO_HELP, NUTHandle);

        NWSAppendCommentField(i, 58, MSG("Hemisphere : ", 686), NUTHandle);
        mfctl2 = NWSInitMenuField(InxMSG("Ground Longitude Hemisphere", 687), 10
, 40, GroundLongHemHandler, NUTHandle);
        NWSAppendToMenuField(mfctl2, InxMSG("West", 688), 0, NUTHandle);
        NWSAppendToMenuField(mfctl2, InxMSG("East", 689), 1, NUTHandle);
        NWSAppendMenuField(i, 71, NORMAL_FIELD, (int *)&sat.cityEastFlag, mfctl2
, NULL, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Ground Latitude  degrees : ", 690), NUT
Handle);
        NWSAppendIntegerField(i, 30, NORMAL_FIELD, (int *)&sat.cityLatDegrees, 1
, 180, F_NO_HELP, NUTHandle);

        NWSAppendCommentField(i, 40, MSG("minutes : ", 691), NUTHandle);
        NWSAppendIntegerField(i, 50, NORMAL_FIELD, (int *)&sat.cityLatMinutes, 1
, 180, F_NO_HELP, NUTHandle);

        NWSAppendCommentField(i, 58, MSG("Hemisphere : ", 692), NUTHandle);
        mfctl3 = NWSInitMenuField(InxMSG("Ground Latitude Hemisphere", 693), 10,
 40, GroundLatHemHandler, NUTHandle);
        NWSAppendToMenuField(mfctl3, InxMSG("South", 694), 0, NUTHandle);
        NWSAppendToMenuField(mfctl3, InxMSG("North", 695), 1, NUTHandle);
        NWSAppendMenuField(i, 71, NORMAL_FIELD, (int *)&sat.cityNorthFlag, mfctl
3, NULL, NUTHandle);

        i++;
        NWSAppendHotSpotField(i, 35, NORMAL_FIELD, MSG("COMPUTE NOW", 696),
                                ComputeHotSpot, NUTHandle);

        CalculateDish(&sat, &dish);

        if (sat.horzFlag == 0)
        {
                dish.polarization += 90.0;
                if (dish.polarization > 90.0)
                        dish.polarization -= 90.0;
                if (dish.polarization < -90.0)
                        dish.polarization += 90.0;
        }

        i+=2;

        NWsprintf(elevationStr,    MSG("      Elevation    : %lf", 697), dish.elev
ation);
        NWsprintf(trueAzimuthStr,  MSG("  True Azimuth     : %lf", 698), dish.true
Azimuth);
//      if ((strcmpi(country, MSG("USA", 243))) != 0)
//              strcpy(magAzimuthStr, MSG("Magnetic Azimuth : N/A", 699));
//      else
                NWsprintf(magAzimuthStr,   MSG("Magnetic Azimuth : %lf", 568), d
ish.magAzimuth);
        NWsprintf(polarizationStr, MSG("    Polarization : %lf", 700), dish.pola
rization);

        NWSAppendCommentField(i, 2, elevationStr, NUTHandle);
        i++;
        NWSAppendCommentField(i, 2, trueAzimuthStr, NUTHandle);
        i++;
        NWSAppendCommentField(i, 2, magAzimuthStr, NUTHandle);
        i++;
```

```
        NWSAppendCommentField(i, 2, polarizationStr, NUTHandle);

        i++;

        NWSEditPortalForm(InxMSG("Antenna Pointing Calculations", 701),
                12, 40,
/* center line, column  */
                i, 78,
/* form height, width   */
                F_NOVERIFY, F_NO_HELP,                  /* Control flags, help m
essage */
                NULL,
                NUTHandle);                             /* Confirm message, hand
le */

        NWSDestroyForm(NUTHandle);
        if (NewCalculationFlag != 0)
                goto calculateAgain;

}


void ComputeGetCity(char *region)
{
        DIR *dirCountry, *dirCity;
        char *name;
        LIST    *listPtr = NULL;
        LONG ccode = M_SELECT;
        int rows = 0, cols = 0;
        FILE *fp;
        char szTmp[128];
        char byteStr[8];
        int latDeg, latMin, longDeg, longMin;
        int len, start = 0;
        LONG info;

        NWSStartWait( 0, 0, NUTHandle );        /* DWH change 970131 */
getCitiesLoop:
        rows = cols = 0;
        listPtr = NULL;
        NWSInitList(NUTHandle, NULL);

        dirCountry = opendir(MSG("*.cty", 702));
        if (dirCountry == NULL)
                goto errorNoDir;

        while( (dirCity = readdir(dirCountry)) != NULL )
        {
                name = dirCity->d_name;

                fp = fopen(name, MSG("r", 703));
                if (fp == NULL)
                {
                        closedir(dirCountry);
                        goto errorNoDir;
                }

                len = strlen(region);
                if(fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
                {
                        if ( (strnicmp(region, szTmp, len)) == 0)
                                break;
                }
                fclose(fp);
        }
        if (dirCity == NULL)
        {
                closedir(dirCountry);
                goto errorNoDir;
        }

        while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
        {
                int len = strlen(szTmp) - 2;
                byteStr[2] = szTmp[len--];
                byteStr[1] = szTmp[len--];
                byteStr[3] = 0;
                longMin = atoi(&byteStr[1]);

                byteStr[2] = szTmp[len--];
                byteStr[1] = szTmp[len--];
                byteStr[0] = szTmp[len--];
                longDeg = atoi(byteStr);

                byteStr[2] = szTmp[len--];
                byteStr[1] = szTmp[len--];
                latMin = atoi(&byteStr[1]);

                byteStr[2] = szTmp[len--];
                byteStr[1] = szTmp[len--];
                latDeg = atoi(&byteStr[1]);

                if (longMin > 59)
                {
                        longDeg++;
                        longMin = 0;
                }

                if (latMin > 59)
                {
                        latDeg++;
                        latMin = 0;
                }

                info = ((latDeg & 0xff) << 24) |
                                          ((latMin & 0xff) << 16) |
                                          ((longDeg & 0xff) << 8) |
                                          (longMin & 0xff);

                while(szTmp[len] == ' ')
                        len--;
                if(len > 0)
                {
                        szTmp[len+1] = 0;

                        while(len)
                        {
                                if (szTmp[len] == ' ')
                                        start = len + 1;
                                len--;
                        }
                        if (start > 0)
                        {
                                AppendToList(&szTmp[start], info, &rows, &cols);
                        }
                }
        }
        fclose(fp);

        if (rows == 0)
        {
```

```
                closedir(dirCountry);
                goto errorNoDir;
        }

        NWSEndWait( NUTHandle );        /* DWH change 970131 */
        ccode = NWSList(
                        InxMSG("Choose A City", 704),
                        12, 40,
                        (rows < 16) ? rows : 16,
/* Height */
                        (cols < 18) ? 18 : cols,
/* Width */
                        M_ESCAPE | M_SELECT,
                        &listPtr,
                        NUTHandle, NULL,
                        NULL, NULL);

        if (ccode == M_SELECT)
        {
                info = (LONG)listPtr->otherInfo;
                strcpy(szTmp, listPtr->text);
                NWSDestroyList(NUTHandle);
                closedir(dirCountry);
                ComputeCity(region, szTmp, info);
                goto getCitiesLoop;
        }

        NWSDestroyList(NUTHandle);
        closedir(dirCountry);
        return;

errorNoDir:
        NWSEndWait( NUTHandle );        /* DWH change 970131 */
        NWSDestroyList(NUTHandle);
        NWSAlert(12, 40, NUTHandle, InxMSG("Unable to access Cities file", 705))
;
        return;

}

void ComputeGetRegion(char *country)
{
        DIR *dirCountry, *dirCity;
        char *name, *end;
        LIST    *listPtr = NULL;
        LONG ccode = M_SELECT;
        int rows = 0, cols = 0;
        char path[64];
        FILE *fp;
        char szTmp[128];
        LONG header;

        NWSStartWait( 0, 0, NUTHandle );        /* DWH change 970131 */
getRegionsLoop:
        rows = cols = 0;
        listPtr = NULL;
        NWSInitList(NUTHandle, NULL);
        SetCurrentNameSpace(DOSNameSpace);
        NWsprintf(path, MSG("SYS:DIRECPC\\DB\\%s.cou", 706), country);
  if (chdir(path))
                goto errorNoDir;

        dirCountry = opendir(MSG("*.cty", 707));
        if (dirCountry == NULL)
                goto errorNoDir;
        while( (dirCity = readdir(dirCountry)) != NULL )
        {
                name = dirCity->d_name;

                fp = fopen(name, MSG("r", 708));
                if (fp == NULL)
                {
                        closedir(dirCountry);
                        goto errorNoDir;
                }

                if(fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
                {
                        end = &szTmp[strlen(szTmp) - 2];
                        while( (*end == ' ') && (end != szTmp) )
                                end--;
                        end++;
                        *end = 0;
                        AppendToList(szTmp, 0, &rows, &cols);
                }
                fclose(fp);
        }

        if (rows == 0)
        {
                closedir(dirCountry);
                goto errorNoDir;
        }

        GetRegionName(szTmp);
        if ( (strcmpi(szTmp, MSG("Province", 709))) == 0)
                header = InxMSG("Choose A Province", 710);
        else
                header = InxMSG("Choose A State", 711);

        if (rows == 1)
        {
                NWSEndWait( NUTHandle );        /* DWH change 970131 */
                NWSDestroyList(NUTHandle);
                ComputeGetCity(szTmp);
                closedir(dirCountry);
                return;
        }
        else
        {
                NWSEndWait( NUTHandle );        /* DWH change 970131 */
                ccode = NWSList(
                                header,
                                12, 40,
                                (rows < 16) ? rows : 16,
/* Height */
                                (cols < 18) ? 18 : cols,
/* Width */
                                M_ESCAPE | M_SELECT,
                                &listPtr,
                                NUTHandle, NULL,
                                NULL, NULL);
        }

        if (ccode == M_SELECT)
        {
                strcpy(szTmp, listPtr->text);
                NWSDestroyList(NUTHandle);
                closedir(dirCountry);
                ComputeGetCity(szTmp);
                goto getRegionsLoop;
```

```
        }

        NWSDestroyList(NUTHandle);

        closedir(dirCountry);
        return;

errorNoDir:
        NWSEndWait( NUTHandle );        /* DWH change 970131 */
        NWSDestroyList(NUTHandle);
        NWSAlert(12, 40, NUTHandle, InxMSG("Unable to locate Country files in Co
untry directory %s.cou", 569), country);
        return;

}


void ComputeCoordinates(void)
{
        DIR *dirDB, *dirCountry;
        char *name, *dot;
        LIST    *listPtr = NULL;
        LONG ccode = M_SELECT;
        int rows = 0, cols = 0;

getCountriesLoop:
        rows = cols = 0;
        listPtr = NULL;
        NWSInitList(NUTHandle, NULL);
        SetCurrentNameSpace(DOSNameSpace);
  if (chdir(MSG("SYS:DIRECPC\\DB", 570)))
                goto errorNoDir;

        dirDB = opendir(MSG("*.cou", 571));
        if (dirDB == NULL)
                goto errorNoDir;

        while( (dirCountry = readdir(dirDB)) != NULL )
        {
                name = dirCountry->d_name;
                if ((dot = strchr(name, '.')) != NULL)
                        *dot = 0;       /* we don't want the extension */
                AppendToList(name, 0, &rows, &cols);
        }

        if (rows == 0)
        {
                closedir(dirDB);
                goto errorNoDir;
        }

        ccode = NWSList(
                        InxMSG("Choose A Country", 572),
                        12, 40,
                        rows,                                   /* Height */
                        18,                                     /* Width */
                        M_ESCAPE | M_SELECT,
                        &listPtr,
                        NUTHandle, NULL,
                        NULL, NULL);

        if (ccode == M_SELECT)
        {
                if (NWSPushList(NUTHandle) != 0)
                {
                        name = listPtr->text;
                        ComputeGetRegion(name);
                        NWSPopList(NUTHandle);
                }
                NWSDestroyList(NUTHandle);
                closedir(dirDB);
                goto getCountriesLoop;
        }

        NWSDestroyList(NUTHandle);

        closedir(dirDB);
        return;

errorNoDir:
        NWSDestroyList(NUTHandle);
        NWSAlert(12, 40, NUTHandle, InxMSG("Unable to locate Country directories
 in SYS:DIRECPC\\DB", 573));
        return;

}

void DPCPointing(void)
{
        LIST    *listPtr = NULL;
        LONG ccode = M_SELECT;

        NWSInitList(NUTHandle, NULL);

        NWSAppendToList(MSG("Antenna Pointing Calculations", 574), (void *)1, NU
THandle);
        NWSAppendToList(MSG("Signal Strength Meter", 575), (void *)2, NUTHandle)
;
        while (ccode != M_ESCAPE)
        {
                ccode = NWSList(
                                InxMSG("Dish Pointing", 576),
                                12, 40,
                                2,                                      /* Heigh
t */
                                30,                                     /* Width
 */
                                M_ESCAPE | M_SELECT,
                                &listPtr,
                                NUTHandle, NULL,
                                NULL, NULL);

                if (ccode == M_SELECT)
                {
                        if (NWSPushList(NUTHandle) != 0)
                        {
                                switch ((int)listPtr->otherInfo)
                                {
                                        case 1:
                                                ComputeCoordinates();
                                                break;

                                        case 2:
                                                SignalMeter();
                                                break;

                                        default:
                                                break;
                                }
                                NWSPopList(NUTHandle);
                        }
                }
```

```
        }
        NWSDestroyList(NUTHandle);
}


void UpdateAdapterInformation(LONG portal)
{
        PCB             *portalPtr;
        int                             line, count, col;
        MUXdacau_t      *dptr;
        BYTE                    string[80];
        char                    serialNumber[10];

        NWSGetPCB( &portalPtr, portal, NUTHandle );

        /* do generic stuff */

        line = 0;

        DIOGetSN(serialNumber);
        NWSShowPortalLine
        (
                line++,
                GblDataCol,
                serialNumber,
                CStrLen(serialNumber),
                portalPtr
        );

        NWSShowPortalLine
        (
                line++,
                GblDataCol,
                SiteID,
                CStrLen(SiteID),
                portalPtr
        );

        NWsprintf(string, MSG("%s", 503), CASDBdacau.entries == 0 ? MSG("FALSE",
 474) : "TRUE ");
        NWSShowPortalLine
        (
                line++,
                GblDataCol,
                string,
                CStrLen(string),
                portalPtr
        );

        count = CASDBdacau.entries;
        NWsprintf( string, MSG("%d", 495), count );
        NWSShowPortalLine
        (
                line++,
                GblDataCol,
                string,
                CStrLen(string),
                portalPtr
        );

        dptr = (MUXdacau_t *)CASDBdacau.p_buffer;
        while (count)
        {
                line++;
                if (line > (portalPtr->virtualHeight - 1))
                        break;
                for (col = 8; col <= 64 ; col+=16, count-- )
                {
                        if (!count)
                                break;

                        NWsprintf( string, MSG("%2.2X%2.2X%2.2X%2.2X", 504),
                                dptr->groupid.i[2], dptr->groupid.i[1],
                                dptr->groupid.i[0], dptr->version);

                        NWSShowPortalLine(
                                line,
                                col,
                                string,
                                CStrLen(string),
                                portalPtr
                        );

                        dptr++;
                }
        }

        NWSUpdatePortal( portalPtr );

}

void DisplayAdapterInfo(void)
{
        int             line, len;
        BYTE    oldPortal;
        PCB    *portalPtr;
        BYTE    string[80];

        line = 5 + 3;   /* Site ID, S/N, Key Status, Number of Communities,
                                                        Current Communities: ,
extra community lines */
        line += (CASDBdacau.entries / 4);

        oldPortal = NUTHandle->currentPortal;
        NWSDeselectPortal( NUTHandle );
        BackgroundPortal = NWSCreatePortal
        (
                1       /* gblUPFTopLine */,
                1 /* gblUPFLeftCol */,
                ((line + 4) > (ScreenHeight - 3)) ? ScreenHeight - 3 : line + 4,
 /* gblUPFHeight + gblLPFHeight */
                ScreenWidth - 2 /* gblUPFWidth */,
                line,
                ( ScreenWidth - 4 /* gblUPFWidth - 2 */ ),
                TRUE,
                MSG("DPC Adapter Information", 502),
                VNORMAL,
                SINGLE,
                VINTENSE,
                CURSOR_OFF,
                VIRTUAL,
                NUTHandle
        );
        if ( BackgroundPortal > MAXPORTALS )
        {
                NWSSelectPortal( oldPortal, NUTHandle );
                return;
        }
        NWSGetPCB( &portalPtr, BackgroundPortal, NUTHandle );
        portalPtr->showScrollBars = SHOW_VERTICAL_SCROLL_BAR;
        portalPtr->showScrollBars |= TEXT_SENSITIVE_SCROLL_BARS;
        portalPtr->verticalScroll = SCROLL_ON;
```

```
        NWSClearPortal( portalPtr );

/*
 * Start filling in the static portal lines.
 */

        line = 0;
        NWsprintf(string, MSG("Serial Number          :     ", 496));
        len = CStrLen( string );
        NWSShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

        NWsprintf(string, MSG("Site ID                :     ", 497));
        len = CStrLen( string );
        NWSShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

        NWsprintf(string, MSG("Have Keys Status       : ", 498));
        len = CStrLen( string );
        NWSShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

        NWsprintf(string, MSG("Number Of Communites : ", 499));
        len = CStrLen( string );
        NWSShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

        NWsprintf(string, MSG("Communities : ", 500));
        len = CStrLen( string );
        NWSShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

        GblDataCol = BORDER_WIDTH + 24;

        UpdateAdapterInformation(BackgroundPortal);
        BackgroundFuncPtr = UpdateAdapterInformation;
        HandleScrollablePortal(portalPtr);
        BackgroundFuncPtr = NULL;
        NWSDestroyPortal( BackgroundPortal, NUTHandle );
        NWSSelectPortal( oldPortal, NUTHandle );

}


void DisplayMLIDStats(void)
{
        int             line;
        int             count;
        int             numberOfGenerics;
        int             len;
        int             promptMax;
        struct DriverStatsStructure *stats;
        struct DriverConfigurationStructure *config;
        ProtocolNodeStructure *protocol;
        CustVars                *customPtr;
        BYTE    *customStrings, *ptr;
        BYTE    oldPortal;
        BYTE    name[128], *namePtr;
        PCB             *portalPtr;
        BYTE    string[80];

        GblDataCol = ( ScreenWidth - 4 ) - BORDER_WIDTH - STATS_DATA_WIDTH;
        if (DPCGetMLIDStats(&stats))
        {
                return;
        }

        DIOGetMLIDConfig(&config);

/*
 * Build up the LAN name followed by its I/O resources
 * to be used as the portals header.
 */

        if (config->DLogicalName[0] != 0)
                NWsprintf(name, MSG("%s [%S", 270), config->DLogicalName, config
->DShortName);
        else
                NWsprintf(name, MSG("[%S", 271), config->DShortName);

        if (config->DIOPortsAndRanges[1] > 0)
        {
                namePtr = name + CStrLen(name);
                NWsprintf(namePtr, MSG(" port=%X", 272), config->DIOPortsAndRang
es[0]);
        }
        if (config->DIOPortsAndRanges[3] > 0)
        {
                namePtr = name + CStrLen(name);
                NWsprintf(namePtr, MSG(" port=%X", 273), config->DIOPortsAndRang
es[2]);
        }
        if (config->DMemoryDecodeAndLength[0].LANMemoryAddress > 0 )
        {
                namePtr = name + CStrLen(name);
                NWsprintf(namePtr, MSG(" memory=%X", 274),
                                config->DMemoryDecodeAndLength[0].LANMemoryAddre
ss);
        }
        if (config->DMemoryDecodeAndLength[1].LANMemoryAddress > 0 )
        {
                namePtr = name + CStrLen(name);
                NWsprintf(namePtr, MSG(" memory=%X", 275),
                                config->DMemoryDecodeAndLength[1].LANMemoryAddre
ss);
        }
        if (config->DIntLine[0] != 0xff)
        {
                namePtr = name + CStrLen(name);
                NWsprintf(namePtr, MSG(" int=%X", 276), config->DIntLine[0]);
        }
        if (config->DIntLine[1] != 0xff)
        {
                namePtr = name + CStrLen(name);
                NWsprintf(namePtr, MSG(" int=%X", 277), config->DIntLine[1]);
        }
        if (config->DDMALine[0] != 0xff)
        {
                namePtr = name + CStrLen(name);
                NWsprintf(namePtr, MSG(" dma=%X", 278), config->DDMALine[0]);
        }
        if (config->DDMALine[1] != 0xff)
        {
                namePtr = name + CStrLen(name);
                NWsprintf(namePtr, MSG(" dma=%X", 279), config->DDMALine[1]);
        }
        if (config->DMediaType != NULL)
        {
                namePtr = name + CStrLen(name);
                NWsprintf(namePtr, MSG(" frame=%S", 280), config->DMediaType);
        }
        namePtr = name + CStrLen(name);
        NWsprintf(namePtr, MSG("]", 281));

/*
 * Before we can create the portal, we must add up the number number
 * of lines we'll need, which will be bigger than the window.
```

```
 */

        line = 3;                       /* lines for version, node address, and
protocol header */


        /* Add up the number of protocols bound to this board */

        protocol = MLIDProtocolListByBoard( DIOBoard );
        while (protocol != NULL)
        {

                ++line;
                protocol = (ProtocolNodeStructure *)protocol->ProtocolBoardLink;
        }

        /* Add in the number of generic statistics */

        line += 2;      /* Blank line and Generic statistics header */

        numberOfGenerics = stats->GenericVariableCount;
        line += numberOfGenerics;

        promptMax = 0;
        for (count = 0; count < numberOfGenerics ; count++ )
        {
                len = CStrLen( GetMsg(GenericDescriptionTable[count]) );
                if (promptMax < len)
                {
                        promptMax = len;
                }
        }

        /* Add in the number of custom statistics */
        line += 2;              /* Blank line and Custom statistics header */

        customPtr = (CustVars *)(&stats->CustomVariableCount);
        customStrings = (BYTE *)(customPtr->CustomVariable[customPtr->CustomVari
ableCount]);
        customStrings += sizeof(WORD);

        line += customPtr->CustomVariableCount;


        /* Check lengths of custom strings */

        ptr = customStrings;    /* temp pointer to walk down custom pro  /
        for ( count = 0; count < customPtr->CustomVariableCount; count++
        {
                len = CStrLen( ptr );
                if ( promptMax < len )
                        promptMax = len;
                while ( *( ptr++ ) )
                        ;       /* find the next string */
        }

        line += 1;      /* add a blank line for the bottom */

        if ( promptMax < 46 )
                promptMax = 64;         /* minimum portal width */
        else if ( promptMax > 60 )
                promptMax = 76;         /* maximum portal width */
        else
                promptMax += 16;        /* custom portal width within range */

        /* build the portal */
        if ( line < 8 )
                line = 8;                /* must meet the minimum portal size */

        oldPortal = NUTHandle->currentPortal;
        NWSDeselectPortal( NUTHandle );
        BackgroundPortal = NWSCreatePortal
        (
                1       /* gblUPFTopLine */,
                1 /* gblUPFLeftCol */,
                ScreenHeight - 3 /* gblUPFHeight + gblLPFHeight */,
                ScreenWidth - 2 /* gblUPFWidth */,
                line,
                ( ScreenWidth - 4 /* gblUPFWidth - 2 */ ),
                TRUE,
                name,
                VNORMAL,
                SINGLE,
                VINTENSE,
                CURSOR_OFF,
                VIRTUAL,
                NUTHandle
        );
        if ( BackgroundPortal > MAXPORTALS )
        {
                NWSSelectPortal( oldPortal, NUTHandle );
                return;
        }
        NWSGetPCB( &portalPtr, BackgroundPortal, NUTHandle );
        portalPtr->showScrollBars = SHOW_VERTICAL_SCROLL_BAR;
        portalPtr->showScrollBars |= TEXT_SENSITIVE_SCROLL_BARS;
        portalPtr->verticalScroll = SCROLL_ON;
        NWSClearPortal( portalPtr );

/*
 * Start filling in the static portal lines.
 */

        /* Fill in the MLID version */

        line = 0;
        NWsprintf(string, MSG("Version %d.%d", 282), config->DMLID_MajorVersion,
 config->DMLID_MinorVersion);
        len = CStrLen( string );
        NWSShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

        /* Fill in the node address */

        NWsprintf(string, MSG("Node Address: %X%4.4X", 283),
                        GET_HILO_LONG( (LONG *)&config->DNodeAddress[0]),
                        GET_HILO_WORD( (WORD *)&config->DNodeAddress[4]));
        len = CStrLen( string );
        NWSShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

        /* protocols */

        NWSShowPortalLine( line++, BORDER_WIDTH, MSG("Protocols:", 284),
                        CStrLen( MSG("Protocols:", 287) ), portalPtr );
        protocol = MLIDProtocolListByBoard[ DIOBoard ];
        while ( protocol != NULL )
        {
//              GetProtocolNameTableEntry( protocol->ProtocolNumber, string );
                CMovB(ProtocolNameTable[protocol->ProtocolNumber], string, 16);
                len = string[ 0 ];
                string[ len + 1 ] = NULL;
                NWSShowPortalLine
```

```
                (
                        line++,
                        BORDER_WIDTH + INDENT_WIDTH,
                        &string[ 1 ],
                        len,
                        portalPtr
                );
                if ( LStrCmp( string, MSG("\x003IPX", 285) ) == 0 )
                {
                        LONG tmpLong;

                        /* network address */

                        tmpLong = IPXNetNumberTable[ DIOBoard ];
                        NWsprintf( string, MSG("Network address:  ", 286), GET_H
ILO_LONG( &tmpLong ));
                        len = CStrLen( string );
                        NWSShowPortalLine
                        (
                                line++,
                                BORDER_WIDTH + ( 2 * INDENT_WIDTH ),
                                string,
                                len,
                                portalPtr
                        );
                }
                protocol = (ProtocolNodeStructure *)protocol->ProtocolBoardLink;
        }

        /* Generic Statistics */

        line++;
        len = CStrLen( MSG("Generic statistics", 288) );
        NWSShowPortalLine( line++, BORDER_WIDTH, MSG("Generic statistics", 289),
 len, portalPtr );
        GenericLineStart = line;
        promptMax -= 16;

        for ( count = 0; count < numberOfGenerics; count++ )
        {
                ptr = GetMsg(GenericDescriptionTable[ count ]);
                len = CStrLen( ptr );
                if ( len > promptMax )
                        len = promptMax;
                NWSShowPortalLine
                (
                        line++,
                        BORDER_WIDTH + INDENT_WIDTH,
                        ptr,
                        len,
                        portalPtr
                );
        }

        /* custom stats */

        line++;
        len = CStrLen( MSG("Custom statistics", 227) );
        NWSShowPortalLine( line++, BORDER_WIDTH, MSG("Custom statistics", 292),
len, portalPtr );

        for ( count = 0; count < customPtr->CustomVariableCount; count++ )
        {
                NWSShowPortalLine
                (
                        line++,
                                BORDER_WIDTH + INDENT_WIDTH,
                                customStrings,
                                CStrLen( customStrings ),
                                portalPtr
                        );
                        while ( *( customStrings++ ) )
                                ;       /* find the next string */
                }

                UpdateStatsInformation(BackgroundPortal);
                BackgroundFuncPtr = UpdateStatsInformation;
                HandleScrollablePortal(portalPtr);
                BackgroundFuncPtr = NULL;
                NWSDestroyPortal( BackgroundPortal, NUTHandle );
                NWSSelectPortal( oldPortal, NUTHandle );


}

LONG    DisconnectRoutine(FIELD *fp, int key, int *changed, NUTInfo *handle)
{
        fp = fp;
        key = key;
        changed = changed;
        handle = handle;

        DloEndConn();

        return(K_NORMAL);

}

LONG    DialInternetRoutine(FIELD *fp, int key, int *changed, NUTInfo *handle)
{
        fp = fp;
        key = key;
        changed = changed;
        handle = handle;

        DloStartConn(DLO_INET_TIMEOUT);

        return(K_NORMAL);

}

LONG    DialPDRoutine(FIELD *fp, int key, int *changed, NUTInfo *handle)
{
        fp = fp;
        key = key;
        changed = changed;
        handle = handle;

        DloStartConn(DLO_PACKAGE_TIMEOUT);

        return(K_NORMAL);

}

LONG    SendModemRoutine(FIELD *fp, int key, int *changed, NUTInfo *handle)
{
        BYTE sendStr[82];
        int ccode;
        LONG len;
        fp = fp;
        key = key;
```

```
        changed = changed;
        handle = handle;

        sendStr[0] = 0;

        if (!NWSPushList(NUTHandle))
                return(K_NORMAL);

        ccode = NWSEditString(
                12, 40,
/* center line, column          */
                1, 40,
/* edit height, width           */
                InxMSG("Modem Send Editor", 587),       /* header
*/
                InxMSG("Send : ", 588),                 /* prompt
*/
                (BYTE*)&sendStr, 80,
/* buffer, max len                      */
                EF_ANY, NUTHandle,                              /* type     le
*/
                NULL, NULL,
/* insert Proc, action Proc*/
                MSG("a..z0..9A..Z-_", 589));
        if ( ccode & E_ESCAPE )
        {
                /* Start Change by DWH 961115 */
                NWSPopList(NUTHandle);
                /* End change by DWH 961115 */
                return(K_NORMAL);
        }

        if ( (len = CStrLen(sendStr)) )
        {
                DloSend(sendStr, len, DLO_INET_TIMEOUT);
        }
        DloSend(MSG("\r", 609), 1, DLO_INET_TIMEOUT);

        NWSPopList(NUTHandle);
        return(K_NORMAL);

}


void ModemControl(void)
{
        int i;

        NWSInitForm(NUTHandle);

        i = 0;
        NWSAppendHotSpotField(i, 15-(20/2), NORMAL_FIELD, MSG("Disconnect the Mo
dem", 547),
                                        DisconnectRoutine, NUTHandle);
        i++;
        NWSAppendHotSpotField(i, 15-(18/2), NORMAL_FIELD, MSG("Dial the Internet
", 549),
                                        DialInternetRoutine, NUTHandle);

        i++;
        NWSAppendHotSpotField(i, 15-(26/2), NORMAL_FIELD, MSG("Dial the Package
Delivery", 590),
                                        DialPDRoutine, NUTHandle);

        i++;
        NWSAppendHotSpotField(i, 15-(18/2), NORMAL_FIELD, MSG("Send to the Modem
", 551),
                                    SendModemRoutine, NUTHandle);

        i++;

        NWSEditPortalForm(InxMSG("Modem Control Options", 552),
                10, 30,
/* center line, column  */
                i, 30,
/* form height, width   */
                F_NOVERIFY, F_NO_HELP,                  /* Control flags, help m
essage */
                InxMSG("Save Changes?", 585),
                NUTHandle);                             /* Confirm message, hand
le */

        NWSDestroyForm(NUTHandle);
}

/*********************************************************************
*
*       MainOptionsHandler(void)
*
*       Description:
*               This routine is where the main agent thread lives.
*               It initiates the NUT screen, waits for the DPC MLID
*               to become active, and wait for user commands.
*
*       Input:
*               nothing
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
*********************************************************************/

void    MainOptionsHandler()
{
        int     choice, prevChoice, i;
        int     exitFlag = FALSE;
        LIST    *defaultList;
        LONG type;
        BYTE value;
        int     countdown = MAX_COUNTDOWN;
        LONG mainPortal;
#if DRIVER_IO
        LONG removedCount;
#else
        void (*ControlEntryPoint) ()= NULL;
        LONG    board;
        struct DriverConfigurationStructure *config;
        char *configName;
#endif

        /*
        * Clear the screen by creating huge portal with VNORMAL attribute.
        */

        GetScreenSize(&ScreenHeight, &ScreenWidth);
        ScreenHeight = ScreenHeight - NUTHandle->headerHeight;
        mainPortal = NWSCreatePortal(
                NUTHandle->headerHeight, 0,                 /* line, column
*/
```

```
                ScreenHeight, ScreenWidth,                              /* frame height/
width           */
                ScreenHeight, ScreenWidth,                              /* virtual heigh
t/width         */
                SAVE, NULL,
/* Save flag, header text               */
                VNORMAL, NOBORDER,                                      /* heade
r attr, border type     */
                VNORMAL, CURSOR_OFF,                                    /* borde
r attr, cursor flag     */
                VIRTUAL, NUTHandle);                                    /* direc
t flag, handle                  */

        if (mainPortal >= MAXPORTALS)
                return;

#if DRIVER_IO
LookForAdapter:
#endif

        while (NWSKeyStatus(NUTHandle))
                NWSGetKey(&type, &value, NUTHandle);

        NWSUpdatePortal( NUTHandle->portal[mainPortal] );

        /*
        * Display our server name in an info portal at the top of the screen.
        */

        UpdateHelpPortal();

        /*
        * Lets look for a DPC adapter.
        */

        StartWaitWithMessage(ScreenHeight/2, ScreenWidth/2, NUTHandle, InxMSG("
Waiting for DPC adapter to load", 143));
        DPCName[0] = 3;
#if DRIVER_IO

        while(DIORegisterWithAdapter(DPCName) && exitFlag == FALSE)
        {
                delay(500);
                if (NWSKeyStatus(NUTHandle))
                {
                        NWSGetKey(&type, &value, NUTHandle);
                        if ((type == K_ESCAPE) || (type == K_AF10))
                        {
                                NWSEndWait(NUTHandle);
                                return;
                        }
                }
                if (--countdown == 0)
                {
                        Spin(NUTHandle);
                        countdown = MAX_COUNTDOWN;
                }
        }
        if (exitFlag)
                return;
        removedCount = DIORemovedCount;
        choice = prevChoice = PackageDelivery ? 1 : 2;
#else
        while(1)
        {
                for(board = 0; board < NumberOfLANs; board++)
                {
                        CLSLGetMLIDControlEntry(board, (void(*)())&ControlEntryP
oint);
                        if (ControlEntryPoint)
                        {
                                config = (struct DriverConfigurationStructure *)
                                                                        CommandMlid(boar
d, 0, (LONG)ControlEntryPoint);
                                if (config)
                                {
                                        configName = config->DShortName;
                                        if (!CStrCmp(configName, DPCName))
                                                goto FoundDPCBoardNumber;
                                }
                        }
                }


                delay(500);
                if (NWSKeyStatus(NUTHandle))
                {
                        NWSGetKey(&type, &value, NUTHandle);
                        if ((type == K_ESCAPE) || (type == K_AF10))
                        {
                                NWSEndWait(NUTHandle);
                                return;
                        }
                }
                if (--countdown == 0)
                {
                        Spin(NUTHandle);
                        countdown = MAX_COUNTDOWN;
                }
        }
#endif
        /*
         * OK. We have a DPC MLID. Lets set up the main menu and
         *      wait for the user to do something.
         */

#if !DRIVER_IO
FoundDPCBoardNumber:
#endif

        NWSEndWait(NUTHandle);
        NWSEnableInterruptKey(K_AF10, ExitHandler, NUTHandle);

        /*
        * Initialize the main options menu.
        */

        NWSInitDhList(NUTHandle, Free); /* Don't sort menu items. */

        NWSSetDynamicMessage(DYNAMIC_MESSAGE_ONE,
                (BYTE *)"Package Delivery       ", &NUTHandle->messages);
        NWSSetDynamicMessage(DYNAMIC_MESSAGE_TWO,
                MSG("Display MLID Stats   ", 102), &NUTHandle->messages);
        NWSSetDynamicMessage(DYNAMIC_MESSAGE_THREE,
                MSG("DPC Configuration  ", 95), &NUTHandle->messages);
        NWSSetDynamicMessage(DYNAMIC_MESSAGE_FOUR,
                MSG("Dish Pointing         ", 344), &NUTHandle->messages);
        NWSSetDynamicMessage(DYNAMIC_MESSAGE_FIVE,
                MSG("Adapter Information    ", 150), &NUTHandle->messages);
        NWSSetDynamicMessage(DYNAMIC_MESSAGE_SIX,
                MSG("Modem Control          ", 501), &NUTHandle->messages);
        NWSSetDynamicMessage(DYNAMIC_MESSAGE_SEVEN,
```

```
                MSG("Exit DPCAGENT          ", 586), &NUTHandle->messages);

        if (PackageDelivery)
                NWSAppendToMenu(DYNAMIC_MESSAGE_ONE, 1, NUTHandle);
        NWSAppendToMenu(DYNAMIC_MESSAGE_TWO, 2, NUTHandle);
        NWSAppendToMenu(DYNAMIC_MESSAGE_THREE, 3, NUTHandle);
        NWSAppendToMenu(DYNAMIC_MESSAGE_FOUR, 4, NUTHandle);
        NWSAppendToMenu(DYNAMIC_MESSAGE_FIVE, 5, NUTHandle);
        NWSAppendToMenu(DYNAMIC_MESSAGE_SIX, 6, NUTHandle);
        NWSAppendToMenu(DYNAMIC_MESSAGE_SEVEN, 7, NUTHandle);
        while (exitFlag == FALSE)
        {
                prevChoice = choice;

                /* Set defaultList to previous choice */
                defaultList = NWSGetListHead(NUTHandle);
                if (!PackageDelivery)
                        choice -= 1;
                for( i = choice - 1; i; i--)
                        defaultList = defaultList->next;

                choice = NWSMenu(InxMSG("DPCAGENT Options", 151),
                        10, 40, defaultList, NULL, NUTHandle, NULL);
                if (removedCount != DIORemovedCount)
                {
                        NWSDestroyMenu(NUTHandle);
                        NWSClearPortal( NUTHandle->portal[mainPortal] );
                        goto LookForAdapter;
                }
                switch (choice) {
                case 1:
                  if (NWSPushList(NUTHandle)) {
                    DisplayPDInterface();
                    NWSPopList(NUTHandle);
                  }
                  break;

                case 2:
                  /* Display MLID Stats */
                  if (NWSPushList(NUTHandle)) {
                    DisplayMLIDStats();
                    NWSPopList(NUTHandle);
                  }
                  break;

                case 3:
                  /* Modem Configuration */
                  if (NWSPushList(NUTHandle)) {
                    DPCConfiguration();
                    NWSPopList(NUTHandle);
                  }
                  break;

                case 4:
                  /* Signal Strength Meter */
                  if (NWSPushList(NUTHandle)) {
                    DPCPointing();
                    NWSPopList(NUTHandle);
                  }
                  break;

                case 5:
                  /* Display Adapter Information */
                  if (NWSPushList(NUTHandle)) {
                    DisplayAdapterInfo();
                    NWSPopList(NUTHandle);
                }
                break;

            case 6:
              /* Display Adapter Information */
              if (NWSPushList(NUTHandle)) {
                ModemControl();
                NWSPopList(NUTHandle);
              }
              break;

            default:
              if (NWSConfirm(InxMSG("Exit DPCAGENT?", 152), 0, 0, TRUE, NULL
,
                              NUTHandle, NULL) == TRUE)
                exitFlag = TRUE;
              else
                if (choice != 7)
                  choice = prevChoice;
          }

      }

      NWSDestroyMenu(NUTHandle);
}

/***********************************************************************
 *
 *    DPCAgentMain(void *parm)
 *
 *    Description:
 *                    Main thread. It will initialize the NUT screen and wait
for
 *                    user input.
 *
 *    Input:
 *            parm                                    - ignored
 *
 *    Output:
 *            Nothing
 *
 *    Returns:
 *            Nothing
 *
 ***********************************************************************/

void    DPCAgentMain(void *parm)
{
        parm = parm;

        MainOptionsHandler();

        ReturnResources(1);
        exit(1);
}

/***********************************************************************
 *
 *    main(int argc, char *argv[])
 *
 *    Description:
 *                    Initialization routine.
 *
 *    Input:
 *            Nothing
```

```
 *
 *    Output:
 *            Nothing
 *
 *    Returns:
 *            0 if successfully initialized
 *
 ***********************************************************************/

        LONG ScreenID;
LONG main(int argc, char *argv[])
{
        LONG currentScreen;
        LONG ser[2];
        LONG    ccode;
        int i;

        for (i = 1; i < argc; i++)
        {
                if (ICmpB(argv[i], MSG("-DEBUG", 182), 6) == -1)
                {
                  if ((DebugFlag = strtol(&argv[i][6], 0, 16)) == 0)
                        DebugFlag = TRUE;
                }
        }

        /* Get a handle for allocating a resource tag */
        NLMHandle = (struct LoadDefinitionStructure *)GetNLMHandle();
        if (!NLMHandle)
                return(-1);

        if ( ReturnMessageInformation((LONG)NLMHandle, (BYTE ***)&NLMMessageTabl
e, NULL, NULL, NULL))
                return(-1);


        OSGetCountryInfo( &GblDOSCountryInfo );

        /* Allocate a resource tag to use for memory allocations */
        allocRTag = AllocateResourceTag(NLMHandle, TxtMSG("DPCAGENT Memory", 167
),
                                                        AllocSignature);

        AESTag = AllocateResourceTag(NLMHandle, TxtMSG("DPCAGENT AES", 168),
                                                        AESProcessSignature);

        asyncIOTag = AllocateResourceTag(NLMHandle, TxtMSG("DPCAGENT Async I/O",
169),
                                                        ASYNCIOSignature);

        timerTag = AllocateResourceTag(NLMHandle, TxtMSG("DPCAGENT Delay Timer",
170),
                                                        TimerSignature);


        if (allocRTag == NULL ||
                AESTag == NULL ||
                timerTag == NULL ||
                ASYNCIOSignature == NULL)
        {
                ConsolePrintf(TxtMSG("DPCAGENT: Unable to allocate Resource Tags
.", 171));
                return(-1);
        }

        /* Create a screen for displaying our information */
        currentScreen = GetCurrentScreen();
        SetAutoScreenDestructionMode(TRUE);

        /* Initialize the screen Interface */
        ScreenID = CreateScreen("DPCAgent Utility", AUTO_DESTROY_SCREEN);
        ccode = NWSInitializeNut(InxMSG("DPC AGENT PROGRAM", 172), AGENT_VERSION
,
                SMALL_HEADER, NUT_REVISION_LEVEL, NULL, NULL,
                ScreenID, (LONG)allocRTag, &NUTHandle);
        if (ccode)
        {
                ConsolePrintf(TxtMSG("DPCAGENT: Unable to initialize NUT.", 174)
);
                return(-1);
        }
//      NLMMessageTable = (BYTE **)&(NUTHandle->messages);

        /* Get a connection with the server we're on */
        if (DebugFlag) {
          DisplayScreen(ScreenID);
          SetCurrentScreen(currentScreen);
#ifdef LOG_ECB_ACTIVITY
          if (DebugFlag >= 0x51) {
            DPC_TGID = GetThreadGroupID();
            LogRegisterClient("SYS:DIRECPC/LOG.CFG", 2048, &LogClientHandle);
            LogRegisterEvent("ECB", &LogECBHandle);
          }
#endif /* LOG_ECB_ACTIVITY */
        }
        else {
          DestroyScreen(currentScreen);
        }
#if DEBUG_ALL
        if (OpenScreen(MSG("DPCAGENT Debug Screen", 224), screenTag, &DebugScree
nID))
        {
                ConsolePrintf(MSG("DPCAGENT: Unable to create debug screen.", 22
5));
                return(-1);
        }
#endif

        DPCUpdateConfig();

        InetChangeProtocol();

        DPCSetMaxConnections(ser);

        DPCAgentPID = BeginThread(DPCAgentMain, NULL, NULL, NULL);
        RenameThread(DPCAgentPID, "DPCAgent Main");
        if (PackageDelivery) {
          DPCFilePID = BeginThread(DPCFileMain, NULL, 32 * 1024, NULL);
          RenameThread(DPCFilePID, "DPCAgent PD");
          DPCAccessPID = BeginThread(AccessMain, NULL, NULL, NULL);
          RenameThread(DPCAccessPID, "DPCAgent Access");
        }
        DPCModemPID = BeginThread(DloMain, NULL, NULL, NULL);
        RenameThread(DPCModemPID, "DPCAgent Modem");
        if (DPCMaxConnections) {
          DPCInetPID = BeginThread(InetMain, NULL, NULL, NULL);
          RenameThread(DPCInetPID, "DPCAgent TInet");
        }
        signal(SIGTERM, ReturnResources);
        ExitThread(TSR_THREAD, 0);
}
```

```
/*****************************************************************************
 *
 *      ReturnResources(int sig)
 *
 *      Description:
 *                      Shutdown routine. Returns the modules resources.
 *
 *      Input:
 *              sig                                             - ignored
 *
 *      Output:
 *              Nothing
 *
 *      Returns:
 *              Nothing
 *
 *****************************************************************************/

void ReturnResources(int sig)
{
        int i;
        int countdown = MAX_COUNTDOWN;

        sig = sig;
        ExitingFlag = TRUE;

        if (InReturnResources)
                return;

        InReturnResources = TRUE;

/*
 * Force NUT to escape out of all menus so that it can clean
 * before we call NWSRestoreNUT(NUT has a bug were it will
 * attempt to free up its memory twice(ABEND) if the user
 * leaves the screen a couple of menus in before unloading
 * the application from the command line.
 */

        for(i = 0; i < 4; i++)
                NWSUngetKey(UGK_ESCAPE_KEY, UGK_NORMAL_KEY, NUTHandle);

        StartWaitWithMessage(ScreenHeight/2, ScreenWidth/2, NUTHandle,
                        InxMSG("Waiting for threads to Exit", 226));

        if (AccessAsleep)
                ResumeThread(DPCAccessPID);

        if (InetAsleep)
                ResumeThread(DPCInetPID);

/*
 * Give threads and NUT a chance to execute.
 * Wait 1.5 seconds since signal meter and stats could be sleeping
 * for up to a second.
 */

        delay(1500);

        while(DPCFilePID || DPCModemPID || DPCAccessPID || DPCInetPID) {
          void DPCPDTerminate(void);
          DPCPDTerminate();

          if (AccessAsleep)
            ResumeThread(DPCAccessPID);
          if (InetAsleep)
            ResumeThread(DPCInetPID);

          if (--countdown == 0) {
            Spin(NUTHandle);
            countdown = MAX_COUNTDOWN;
          }
          ThreadSwitchWithDelay();
        }

#ifdef LOG_ECB_ACTIVITY
        if (DebugFlag >= 0x51) {
          LogDeregisterEvent(&LogECBHandle);
          LogDeregisterClient(&LogClientHandle);
        }
#endif /* LOG_ECB_ACTIVITY */

        if (DloCfg.out_protocol == OUT_PPP)
        {
                DisconnectPPP();
        }
        DIODeRegisterAgent();

#if DEBUG_ALL
        CloseScreen(DebugScreenID);
#endif
        NWSEndWait(NUTHandle);
        NWSRestoreNut(NUTHandle);

}
```

```
#include "dpcagent.h"          /* Our header file */

LONG                              AccessChannel = -1;
int                                       WaitingForKeys = FALSE;
/*
 * Access Configuration Variables
 */
BYTE SiteID[9];
WORD CDBVersion = 0;
WORD CDBEthVersion = 0;
BYTE DacauFlag = 0;
LONG DacauTime = 0;
LONG RndDacauTime = 0;
DACAUrequest_t DacauRequest;
CASDBbuffer CASDBpacau;
CASDBbuffer CASDBpeb;
CASDBbuffer CASDBdacau;
CASDBbuffer CASDBecau;

/* Elements tables */
static CDBelement_t Elements[MAXELEMENTS];
static CDBelement_t UpdatedElements[MAXELEMENTS];


BYTE *RcvBuf;
MACrecord_t CASmacs[MAX_MAC_RECORDS];
BYTE AccessAddress[] = {0x03, 0x00, 0x00, 0x00, 0x00, 0x00};
LONG AccessAsleep = FALSE;                          /* Access thread needs t
o wake up flag  */

BYTE                           AccessMessage[1500];
/* Stores current packet                */
/*
 *      ECB linked list varaibles.
 */
static ECB      *AccessECBHead = 0;     /* Take ECBs from here */
static ECB      *AccessECBTail = 0;     /* Put ECBs here */

/*
 * element key used by LroSetAddress.
 */
BYTE MagicKey[] = { 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11 };

/*
 * Address Type Table used by MACbuildAddr().
 */
static unsigned char table[5][2] = {
/*                                  indiv/mult      bypass/norm */
/* Hybrid Internet  */              { MAC_INDIVID,   MAC_NORMAL },
/* CAS individual   */              { MAC_INDIVID,   MAC_BYPASS },
/* Package Delivery */              { MAC_MULTICAST, MAC_NORMAL },
/* Data Feed        */              { MAC_MULTICAST, MAC_NORMAL },
/* BYPASS           */              { MAC_MULTICAST, MAC_BYPASS }
};

BYTE SerialNum[9];
BYTE SerialNumPacked[3];

/*********************************************************************
 *
 *      ReadConfig(void)
 *
 *      Description:
 *                      This routine reads the DPC.CFG file and stores the conte
nts
```

```
 *                      into the appropriate globale variables.
 *
 *      Input:
 *              Nothing
 *
 *      Output:
 *              Nothing
 *
 *      Returns:
 *              Nothing
 *
 *********************************************************************/

static void ReadConfig(void)
{
        int     handle, k;
        BYTE    *mem_ptr;

        for(k = 0; k < MAXELEMENTS; k++)
        {
        Elements[k].in_use = 'N';
      UpdatedElements[k].in_use = 'N';
        }

        handle = open(MSG("SYS:DIRECPC\\DB\\DPC.CFG", 203), O_RDONLY);
        if (handle != -1)
        {
                read(handle, &SiteID, sizeof(SiteID));
                read(handle, &CDBVersion, sizeof(CDBVersion));
                read(handle, &CDBEthVersion, sizeof(CDBEthVersion));
                read(handle, &DacauFlag, sizeof(DacauFlag));
                read(handle, &DacauTime, sizeof(DacauTime));
                read(handle, &DacauRequest, sizeof(DACAUrequest_t));
                read(handle, &CASDBpacau, sizeof(CASDBbuffer));
                read(handle, &CASDBpeb, sizeof(CASDBbuffer));
                read(handle, &CASDBdacau, sizeof(CASDBbuffer));
                read(handle, &CASDBecau, sizeof(CASDBbuffer));
        }
        else
                UpdateFileStatus(MSG("Obtaining Encryption Keys", 204));

        for(k = 0; k < MAX_MAC_RECORDS; k++)
                CASmacs[k].in_use = 'N';

        if(handle != -1)
        {
                close(handle);
        }
        else
        {
                SiteID[0] = '\0';
                CASDBpacau.version = CASDBpeb.version = 0;
                CASDBdacau.version = CASDBecau.version = 0;
                CASDBpacau.entries = CASDBpeb.entries = 0;
                CASDBdacau.entries = CASDBecau.entries = 0;
        }
        CASDBpacau.entry_len = PACAU_LEN;
  CASDBpeb.entry_len = PEB_LEN;
  CASDBdacau.entry_len = DACAU_LEN;
  CASDBecau.entry_len = ECAU_LEN;

        if (DacauFlag)
        {
                UpdateFileStatus(MSG("Retry: Obtaining Encryption Keys", 257));
                WaitingForKeys = TRUE;
        }
        DacauRequest.opcode = DACAU_REQUEST;
        RndDacauTime = time(0) + rand();
}

/************************************************************************
*
*       SaveConfig(void *parm)
*
*       Description:
*                       This routine writes the access structures out to DPC.CFG
 anytime
*                       a change is made to any of the structures.
*
*       Input:
*               Nothing
*
*       Output:
*               Nothing
*
*       Returns:
*               Nothing
*
************************************************************************/

static void SaveConfig(void)
{
        int handle;
        int tmpFlag;

        handle = open(MSG("SYS:DIRECPC\\DB\\DPC.CFG", 206), O_RDWR | O_CREAT, S_
IWRITE | S_IREAD);

        tmpFlag = DacauFlag;
        if (WaitingForKeys)
        {
                DacauFlag = 1;
        }
        /* Write version */
   write(handle, SiteID, sizeof(SiteID));
   write(handle, &CDBVersion, sizeof(CDBVersion));
   write(handle, &CDBEthVersion, sizeof(CDBEthVersion));
   write(handle, &DacauFlag, sizeof(DacauFlag));
   write(handle, &DacauTime, sizeof(DacauTime));
   write(handle, &DacauRequest, sizeof(DACAUrequest_t));

   /* Write Info headers */
   write(handle, &CASDBpacau, sizeof(CASDBbuffer));
   write(handle, &CASDBpeb, sizeof(CASDBbuffer));
   write(handle, &CASDBdacau, sizeof(CASDBbuffer));
   write(handle, &CASDBecau, sizeof(CASDBbuffer));

   /* Write buffers */
   close(handle);

        DacauFlag = tmpFlag;
}

/************************************************************************
*
*       AccessESR(ECB *ecb)
*
*       Description:
*               This routine is called by the MLID interrupt service
*               routine when a packet is received. The ECB is queued
*               and if the Access File thread is sleeping, it is
```

```
*               woken up.
*
*       Input:
*               ecb                             - pointer to the ECB describing
the packet
*
*       Output:
*               nothing
*
*       Returns:
*               0                                       We always keep the ECB
*
************************************************************************/

int     AccessESR(ECB *ecb)
{

        /* Link the ECB to the Tail of the linked list */
        ecb->ECB_NextLink = 0;
        if (AccessECBTail)
          AccessECBTail->ECB_NextLink = ecb;
        AccessECBTail = ecb;
        if (AccessECBHead == 0)
          AccessECBHead = ecb;

        /* Wake up file thread only if we need to */
        if (AccessAsleep)
                ResumeThread(DPCAccessPID);

#ifdef LOG_ECB_ACTIVITY
        if (LogECBHandle) {
          int TGID = SetThreadGroupID(DPC_TGID);
          LogMsg(LogClientHandle, LogECBHandle, FALSE,
                 "ACCESS queue(%08lx)\n", ecb);
          SetThreadGroupID(TGID);
        }
#endif /* LOG_ECB_ACTIVITY */

        return(0);
}

/************************************************************************
*
*       find_peb(ID element_pattern,
*                       char ver_pattern)
*
*       Description:
*                       This routine searches the PEB list to find an entry that
*                       matches the element and version pattern passed in.
*
*       Input:
*               element_pattern                 - 3 byte element to search for
*               ver_pattern                                     - 1 byte version
 to search for
*
*       Output:
*               nothing
*
*       Returns:
*               pointer to peb entry if it was found
*               otherwise its a NULL
*
************************************************************************/

MUXecau_t *find_ecau(ID group_pattern, char ver_pattern)
{
        int i;
  MUXecau_t *p_ecau, *ret = NULL;

  for(i = 0; i < CASDBecau.length; i += ECAU_LEN)
  {
        p_ecau = (MUXecau_t *)(CASDBecau.p_buffer + i);
    if(CCmpB(&p_ecau->groupid, &group_pattern, sizeof(ID)) == -1)
    {
        if(ver_pattern == -1)
        {
        ret = p_ecau;
        break;
      }
                else
                {
                if(ver_pattern == p_ecau->version)
                {
                ret = p_ecau;
                break;
        }
      }
                }
        }
  return(ret);
}


static MUXpeb_t *find_peb(ID element_pattern, char ver_pattern)
{
        unsigned short i, num_addr;
   MUXpeb_t *p_peb, *ret = NULL;

   for(i = 0; i < CASDBpeb.length; i += PEB_LEN)
   {
        p_peb = (MUXpeb_t *)(CASDBpeb.p_buffer + i);
                if(CCmpB(&p_peb->elementid, &element_pattern, sizeof(ID)) == -1)
                {
        if(ver_pattern == -1)
        {
                ret = p_peb;
            break;
                        }
                        else
                        {
                if(ver_pattern == p_peb->version)
                {
                ret = p_peb;
              break;
           }
                }
        }
     num_addr = p_peb->num_addr[0];
     num_addr |= ((unsigned short)p_peb->num_addr[1]) << 8;
     i += num_addr * MAC_LENGTH;
        }
   return(ret);
}

/*
 * Deletes element not only from the CASDB
 * but from adapter as well
 */

/*************************************************************************
 *
 *      del_peb_element(int e_num)
```

```
 *
 *      Description:
 *                      This routine deletes and from the CASDB and from the
 *                      adapter.
 *
 *      Input:
 *              e_num                                   - index of the element t
o delete
 *
 *      Output:
 *              nothing
 *
 *      Returns:
 *              0
 *
*************************************************************************/

static del_peb_element(int e_num)
{
        int k;

        DIODeleteAddress(Elements[e_num].channel, (BYTE *)&Elements[e_num].e_mac
);

        for(k = 0; k < MAX_MAC_RECORDS; k++)
        {
        if(CASmacs[k].in_use == 'Y' &&
                CCmpB(&CASmacs[k].dpc_mac, &Elements[e_num].e_mac, sizeof(MACadd
r_t)) == -1)
      {
        CASmacs[k].in_use = 'N';
         break;
                }
        }
   Elements[e_num].in_use = 'N';
   return(0);
}

/*************************************************************************
 *
 *      replace_peb_element(int num,
 *                      unsigned char new_ver)
 *
 *      Description:
 *                      This routine replaces the version numbers of the element
s
 *                      indexed by num.
 *
 *      Input:
 *              int_num                                 - Index of Eleme
nt to modify
 *              new_ver                                 - new version to
 stuff into Element entry
 *
 *      Output:
 *              nothing
 *
 *      Returns:
 *              0
 *
*************************************************************************/

static replace_peb_element(int num, unsigned char new_ver)
{
        int k;
   for(k = 0; k < MAX_MAC_RECORDS; k++)
   {
        if(CASmacs[k].in_use == 'Y' &&
                CCmpB(&CASmacs[k].dpc_mac, &Elements[num].e_mac, sizeof(MACaddr_
t)) == -1)
        {
        CASmacs[k].dpc_mac.Ver = new_ver;
         break;
                }
        }
        Elements[num].e_mac.Ver = new_ver;
   Elements[num].e_ver = new_ver;
   return 0;
}

/************************************************************************
*
*       find_pacau(ID group_pattern,
*               char ver_pattern)
*
*       Description:
*                       This routine attempts to locate a pacau given a group
*                       pattern.
*
*       Input:
*               group_pattern                   - group pattern to match
*               ver_pattern                             - version portion of the
 pattern
*
*       Output:
*
*       Returns:
*               pointer to pacau if successful
*               Otherwise NULL
*
************************************************************************/

MUXpacau_t *find_pacau(ID group_pattern, char ver_pattern)
{
        int i;
   MUXpacau_t *p_pacau, *ret = NULL;

   for(i = 0; i < CASDBpacau.length; i += PACAU_LEN)
   {
        p_pacau = (MUXpacau_t *)(CASDBpacau.p_buffer + i);
      if((CCmpB(&p_pacau->groupid, &group_pattern, 3)) == -1)
      {
        if(ver_pattern == -1)
        {
                ret = p_pacau;
            break;
         }
        else
        {
                if(ver_pattern == p_pacau->version)
                {
                ret = p_pacau;
                break;
            }
         }
                }
        }
        return(ret);
}

/************************************************************************
```

```
*
*       find_dacau(ID group_pattern,
*               char ver_pattern)
*
*       Description:
*                       This routine attempts to locate a dacau given a group
*                       pattern.
*
*       Input:
*               group_pattern                   - group pattern to match
*               ver_pattern                             - version portion of the
 pattern
*
*       Output:
*
*       Returns:
*               pointer to dacau if successful
*               Otherwise NULL
*
************************************************************************

MUXdacau_t *find_dacau(ID group_pattern, char ver_pattern)
{
        int i;
   MUXdacau_t *p_dacau, *ret = NULL;

   for(i = 0; i < CASDBdacau.length; i += DACAU_LEN)
   {
        p_dacau = (MUXdacau_t *)(CASDBdacau.p_buffer + i);
      if((CCmpB(&p_dacau->groupid, &group_pattern, 3)) == -1)
      {
        if(ver_pattern == -1)
        {
                        ret = p_dacau;
                break;
         }
         else
         {
                if(ver_pattern == p_dacau->version)
                {
                ret = p_dacau;
               break;
                }
                }
        }
        }
   return(ret);
}

/************************************************************************
*
*       reverse_key(BYTE *key)
*
*       Description:
*                       This routine swaps the byte order of the 8 byte
*                       key passed in.
*
*       Input:
*               key                                             - key to reverse
*
*       Output:
*
*       Returns:
*               pointer to pacau if successful
*               Otherwise NULL
*
```

```
****************************************************************************/

void reverse_key(BYTE *key)
{
        unsigned char x;

   x = key[0]; key[0] = key[1]; key[1] = x;
   x = key[2]; key[2] = key[3]; key[3] = x;
   x = key[4]; key[4] = key[5]; key[5] = x;
   x = key[6]; key[6] = key[7]; key[7] = x;
}

/****************************************************************************
*
*       make_element_id(BYTE *e_id,
*                       char *e_id_txt)
*
*       Description:
*                       This routine is called by LroSetAddress to help
*                       build the element id.
*
*       Input:
*               e_id
- element id
*
*       Output:
*               e_id_txt
- element text
*
*       Returns:
*               nothing
*
****************************************************************************/

void make_element_id(BYTE *e_id, char *e_id_txt)
{
        BYTE work[3];
   static char HexChar[] = MSG("0123456789ABCDEF", 208);
   unsigned char Ch, *p;
   int count = 3, i = 0;

   work[0] = e_id[2];
   work[1] = e_id[1];
   work[2] = e_id[0];

   p = work;
   while (count--)
   {
       Ch = *p++;
     e_id_txt[i++] = HexChar[Ch>>4];       /* high nibble */
     e_id_txt[i++] = HexChar[Ch & 0x0f];   /* low nibble */
   }
   e_id_txt[i] = '\0';
}

/****************************************************************************
*
*       int43(LONG id, BYTE *array)
*
*       Description:
*                       This routine is called by MACbuildAddr to convert
*                       an id to its 3 byte equivalent.
*
*       Input:
*               id
- id to convert
```

```
*               array                                           - pointe
r to 3 byte array
*
*       Output:
*               array is filled in
*
*       Returns:
*               0 if successful
*
****************************************************************************/

static int43(LONG id, BYTE *array)
{
  union {
    BYTE b[4];
    LONG w;
  } offset;
  int status = 0;

  offset.w = id;
  if (offset.b[3]==0 && !(offset.b[2] & 0xc0)) {
    array[0] = offset.b[2];
    array[1] = offset.b[1];
    array[2] = offset.b[0];
  }
  else {
    array[0]=array[1]=array[2] = 0;
    status = -1;
  }
  return status;
}

/****************************************************************************
*
*       MACbuildAddr(char *element_txt,
*               int feature,
*               BYTE ver,
*               MACaddr_t *address)
*
*       Description:
*                       This routine is called by LroSetAddress to build a
*                       MAC address out of a file_id and community id.
*
*       Input:
*               element_txt                             - file id array
*               feature                                 - MAC_HI, MAC_CA     M
AC_PKG, MAC_DF
*                                                                     of MAC
_BYPASS_MULTICAST
*               ver                                         - low byte of co
mmunity id
*               address                                 - where store the result
ing address
*
*       Output:
*               address is filled in
*
*       Returns:
*               0 if successful
*
****************************************************************************/

int MACbuildAddr(char *element_txt, int feature, BYTE ver, MACaddr_t *address)
{
        BYTE element[3];
   LONG i;
   int k, status = 0;

   if(feature < 0 || feature > 5)
        return(-1);
        /* Step one */
   sscanf(element_txt, MSG("%lx", 137), &i);
   if((status = int43(i, element)) != 0)
        return(status);
        /* Step two */
   for(k=0; k<3; k++)
   {
        element[k] = element[k] << 2;
      element[k] |= ((k == 2) ? 0x00 : element[k+1]) >> 6;
   }
   /* Step three */
   address->Element[0] = element[2];
   address->Element[1] = element[1];
   address->Element[2] = element[0];
   /* Set Multicast/Individual */
   if(table[feature][0] == MAC_MULTICAST)
        address->Element[0] |= MAC_MULTICAST;
        /* Set Bypass/Normal */
   if(table[feature][1] == MAC_BYPASS)
        address->Element[0] |= MAC_BYPASS;
        /* Set application ID or Version number */
   switch(feature)
   {
        case MAC_HI:
                address->Ver = 0x02;
         break;
     case MAC_CAS_IND:
        address->Ver = 0x01;
         break;
        case MAC_BYPASS_MULTICAST:
        address->Ver = 0x00;
         break;
                default:
                        address->Ver = ver;
         break;
        }
        /* Set reserved field */
   address->Reserved[0] = address->Reserved[1] = 0x00;
   if(feature == MAC_DF)
        address->Element[2] = 0xff;
        return(status);
}

/**********************************************************************
*
*       find_peb_mac(MACaddr_t mac_pattern)
*
*       Description:
*                       Finds the PEB entry which contains the mac address passe
d in.
*
*       Input:
*               mac_pattern                             - MAC address to search
for
*
*       Output:
*               Nothing
*
*       Returns:
*               NULL if no entry found
*               Otherwise its a pointer to the PEB entry
*
```

```
***********************************************************************/

static MUXpeb_t *find_peb_mac(MACaddr_t mac_pattern)
{
        unsigned short i, j, num_addr;
   MUXpeb_t *p_peb, *ret = NULL;

   for(i = 0; i < CASDBpeb.length; i += PEB_LEN)
   {
        p_peb = (MUXpeb_t *)(CASDBpeb.p_buffer + i);
      num_addr = p_peb->num_addr[0];
      num_addr |= ((unsigned short)p_peb->num_addr[1]) << 8;
      for(j = 0; j < num_addr; j++)
      {
        if(CCmpB(&mac_pattern,
                                        (unsigned char *)p_peb + PEB_LEN + j*MAC
_LENGTH, MAC_LENGTH) == -1)
                        {
                ret = p_peb;
            goto peb_mac_exit;
         }
      }
                i += num_addr * MAC_LENGTH;

        }
peb_mac_exit:
        return(ret);
}

/**********************************************************************
*
*       find_element_id(ID id,
*                       BYTE ver)
*
*       Description:
*                       Find the elements index into the Element table.
*
*       Input:
*               id                                                      - 3 byte
 id
*               ver                                             - 1 byte version
*
*       Output:
*               Nothing
*
*       Returns:
*               -1 if not found
*               otherwise its the index
*
***********************************************************************/

static find_element_id(ID id, BYTE ver)
{
        int k;
   int ret = -1;

   for(k = 0; k < MAXELEMENTS; k++)
   {
        if(Elements[k].in_use == 'Y' &&
                CCmpB(&Elements[k].e_id, &id, sizeof(ID)) == -1 &&
           Elements[k].e_ver == ver)
        {
        ret = k;
         break;
                }
        }
```

```
    return ret;
}

/**********************************************************************
 *
 *      hextoi(int c)
 *
 *      Description:
 *                      Converts an ASCII hex character into an integer.
 *
 *      Input:
 *              c                                                 - ASCII
character
 *
 *      Output:
 *              Nothing
 *
 *      Returns:
 *              -1 if not hex character
 *              otherwise its the integer equivalent
 *
 **********************************************************************/

static int hextoi(
        int c)
{
        char digit, lower, upper;
   digit = (c >= '0' && c <= '9');
   lower = (c >= 'a' && c <= 'f');
   upper = (c >= 'A' && c <= 'F');

        if (digit)
        return (c - '0');

        if (lower)
        return (c - 'a' + 10);

        if (upper)
        return (c - 'A' + 10);

        return (-1);
}

/**********************************************************************
 *
 *      pack_mac_addr(BYTE *packed_address,
 *                      int packed_address_len,
 *                      BYTE *address,
 *                      int address_len)
 *
 *      Description:
 *                      Converts a character string containing the mac address i
nto
 *                      packed BCD digits.
 *
 *      Input:
 *              packed_address_len      - length of the packed buffer
 *              address                                 - Hex ASCII string to co
nvert
 *              address_len                             - length of the hex ASCI
I string
 *
 *      Output:
 *              packed_buffer                   - buffer to write packed address
 into.
 *
```

```
 *      Returns:
 *              TRUE if packed successfully
 *
 **********************************************************************/

#define     LEFT        0
#define     RIGHT       1
int pack_mac_addr( BYTE *packed_address, int packed_address_len,
        BYTE *address, int address_len)
{
        BYTE c, side;
        int i, j;

        /*
    * Pack hex digit ascii string in "address" into binary in
    * "packed_address". Return FALSE if unsuccessful. If number of hex
    * digits in "address" cannot fill "packed_address", then
    * "packed_address" is padded to the right with zeros.
    */
    side = LEFT;
    for (i = 0; i < packed_address_len; i++)
        packed_address[i] = 0;

        for (i = 0, j = 0; i < address_len; i++)
    {
        if ((c = hextoi(address[i])) == 0xff)
        return (FALSE);

                if (side == LEFT)
      {
        c = c << 4;
                }

      packed_address[j] |= c;

      if (++side > RIGHT)
      {
        side = LEFT;
         if (++j > packed_address_len)
                return (FALSE);
                }
        }

        return (TRUE);
}

/**********************************************************************
 *
 *      check_df_groups(void)
 *
 *      Description:
 *                      This function is called when new PACAU or DACAU is proce
ssing.
 *                      It checks if there are any changes in groups membership
 *                      to avoid receiving DF elements when membership of this a
dapter
 *                      to the proper DF group has been deleted.
 *
 *      Input:
 *              Nothing
 *
 *      Output:
 *              Nothing
 *
 *      Returns:
 *              0 if check was successful
 *
 ***********************************************************************/

static check_df_groups(void)
{
        int i;
   MUXpacau_t *pacau;
   MUXdacau_t *dacau;
   MUXpeb_t *p_peb;

   for(i = 0; i < MAXELEMENTS; i++) {
     if(Elements[i].in_use == 'N' || Elements[i].packfeed != 'F')
       continue;
     if((p_peb = find_peb(Elements[i].e_id, -1)) == NULL)
       continue;
     pacau = find_pacau(p_peb->groupid, p_peb->version);
     dacau = find_dacau(p_peb->groupid, p_peb->version);

     if(dacau == NULL && pacau == NULL) {
       /* We have no group for this element */
       del_peb_element(i);
     }
   }
   return 0;
}

/***********************************************************************
 *
 *       parse_pacau(BYTE *buf, WORD len)
 *
 *       Description:
 *                       Parse the PACAU packet. This is where we detect that we
 *                       must receive a new key via the modem(packets d version
 *                       will not match CASDBdacau.version.
 *
 *       Input:
 *               buf                             - pointer to the message receive
d
 *               len                             - length of the message received
 *
 *       Output:
 *               Nothing
 *
 *       Returns:
 *               0 if successfully parsed
 *
 ***********************************************************************/

static parse_pacau(BYTE *buf, WORD len)
{
   LONG curr_p_version;
   LONG curr_d_version;
   LONG curr_d_time;
   int ret = 0, k;

   if(strncmp(buf, SiteID, 8) != ESUCCESS) {
     strncpy(SiteID, buf, 8);
     SiteID[8] = 0;
     CDBVersion ++;
     /* New Site ID - reset versions of PACAU, DACAU, ...*/
     CASDBpacau.version = CASDBpeb.version = 0;
     CASDBdacau.version = CASDBecau.version = 0;
     SaveConfig();
   }

   curr_p_version =
```

```
     buf[8] +
     buf[9] * 256UL +
     buf[10] * 256UL * 256UL +
     buf[11] * 256UL * 256UL * 256UL;

   curr_d_version =
     buf[12] +
     buf[13] * 256UL +
     buf[14] * 256UL * 256UL +
     buf[15] * 256UL * 256UL * 256UL;

   curr_d_time =
     buf[16] +
     buf[17] * 256UL +
     buf[18] * 256UL * 256UL +
     buf[19] * 256UL * 256UL * 256UL;

   if(curr_d_version != CASDBdacau.version) {
     /* There are some changes in the DACAU staff */
     /* Build DACAU request */
     DacauFlag = 1;
     srand(time(0));
     DacauTime = curr_d_time;
     RndDacauTime = time(0) + rand();
     CDBVersion++;

     WaitingForKeys = TRUE;
     UpdateFileStatus(MSG("Obtaining Encryption Keys", 138));

     memcpy(&DacauRequest.d_version, buf + 8 + sizeof(VER), sizeof(VER));
#ifdef USE_NEW_MIPS_CODE
     DacauRequest.d_version.i[3] |= 0x80;
#endif

     CASDBdacau.version = curr_d_version;
     SaveConfig();
   }

   if(curr_p_version != CASDBpacau.version) {
     CDBVersion++;
     CASDBpacau.version = curr_p_version;
     CASDBpacau.length = len - PACAU_HEAD_LEN;
     memcpy(CASDBpacau.p_buffer,
            buf + PACAU_HEAD_LEN,
            CASDBpacau.length);
     CASDBpacau.entries = CASDBpacau.length / CASDBpacau.entry_len;
     check_df_groups();
     SaveConfig();
   }
   return ret;
}

/***********************************************************************
 *
 *       parse_ecau(BYTE *buf, WORD len)
 *
 *       Description:
 *                       Parse the ECAU packet. Replace the old entry by the
 *                       new one as long as version doesn't match CASDBecau versi
on.
 *
 *       Input:
 *               buf                             - pointer to the message receive
d
 *               len                             - length of the message received
 *
```

```
*       Output:
*               Nothing
*
*       Returns:
*               0 if successfully parsed
*
****************************************************************************/

static parse_ecau(BYTE *buf, WORD len)
{
   LONG curr_e_version;
   int ret = 0;

   curr_e_version =
                     buf[0] +
         buf[1] * 256UL +
         buf[2] * 256UL * 256UL +
         buf[3] * 256UL * 256UL * 256UL;

        if(curr_e_version != CASDBecau.version)
        {
        CDBVersion++;
      CASDBecau.version = curr_e_version;
      CMovB(buf + ECAU_HEAD_LEN, CASDBecau.p_buffer, len - ECAU_HEAD_LEN);
                CASDBecau.length = len - ECAU_HEAD_LEN;
      CASDBecau.entries = CASDBecau.length / CASDBecau.entry_len;
      SaveConfig();
        }
   return ret;
}

/****************************************************************************
*
*       parse_peb(BYTE *buf, WORD len)
*
*       Description:
*                       Parse the PEB packet. Replace the old entry by the
*                       new one. Check elements and add new addresses to the
*                       MLID and delete old ones.
*
*       Input:
*               buf                             - pointer to the message receive
d
*               len                             - length of the message received
*
*       Output:
*               Nothing
*
*       Returns:
*               0 if successfully parsed
*
****************************************************************************/

static parse_peb(BYTE *buf, WORD len)
{
        int i, k, macs, not_found;
  MUXpeb_t *p_peb;
  unsigned long curr_version;
  int ret = 0;
  MUXpacau_t *pacau;
  MUXdacau_t *dacau;

   curr_version =
                buf[0] +
         buf[1] * 256UL +
         buf[2] * 256UL * 256UL +
```

```
         buf[3] * 256UL * 256UL * 256UL;

        if(curr_version != CASDBpeb.version)
        {
        CDBVersion++;
      CDBEthVersion++;
      CASDBpeb.version = curr_version;
      CMovB(buf + PEB_HEAD_LEN, CASDBpeb.p_buffer, len - PEB_HEAD_LEN);
                CASDBpeb.length = len - PEB_HEAD_LEN;
      CASDBpeb.entries = CASDBpeb.length / CASDBpeb.entry_len;
      /* Walk throught the elements table and check ... */
      /* ... if the element still in the PEB */
      for(i = 0; i < MAXELEMENTS; i++)
      {
        if(Elements[i].in_use == 'N' || Elements[i].packfeed != 'F')
                        continue;
                if((p_peb = find_peb(Elements[i].e_id, -1)) == NULL)
                {
                /* Element no longer valid: Delete. */
            del_peb_element(i);
            continue;
        }
         if(p_peb->version != Elements[i].e_ver)
         {
                /* We have found element but with different version: */
            /* It means, that we somehow miss key update */
            /* Replace inside adapter and in CDB. */
            /* Delete address */

                                DIODeleteAddress(Elements[i].channel, (BYTE *)&E
lements[i].e_mac);

                                /* Replace element in the CDB */
            replace_peb_element(i, p_peb->version);
            /* Trying to Add address */
            /* At first find group for the element */
            pacau = find_pacau(p_peb->groupid, p_peb->version);
                                dacau = find_dacau(p_peb->groupid, p_peb->versio
n);

                                if(dacau != NULL)
              pacau = (MUXpacau_t *)dacau;
                                if(pacau != NULL)
                                {
                                        if (DIOAddGroupAddress(Elements[i].chann
el, (BYTE *)&Elements[i].e_mac,
                                                        (BYTE *)&pacau->g   )

                /* Find a group, Add */
//                                      channelCfg.CfgChannel = Elements[i].chan
nel;
//                                      channelCfg.CfgNumAddresses = 1;
//                                      CMovB(&Elements[i].e_mac, channelCfg.Cfg
Address, 8);
//                                      CMovB(&pacau->g_key, key, 8);
//                                      reverse_key(&key);
//                                      CMovB(key, channelCfg.CfgGroupKey, 8);
//                                      CMovB(&MagicKey, channelCfg.CfgElementKe
y, 8);

//                                      ecb.ECB_StackID = MLID_ADD_ADDRESS;
//                                      ecb.ECB_Fragment[0].FragmentAddress = &c
hannelCfg;
//                                      if (IoctlMlid(FDBboard, &ecb, FDBControl
Entry) != 0)
                                                del_peb_element(i);
                                )
            else
            {
                /* Can't find group for this element */
               /* We are not subscribed on this group any more */
               /* Delete element. */
               del_peb_element(i);
                                )
                        )
         /* Check Ethernet addresses within Element. */
         macs = not_found = 0;
         for(k = 0; k < MAX_MAC_RECORDS; k++)
         {
                if(CASmacs[k].in_use == 'Y' &&
                                CCmpB(&CASmacs[k].dpc_mac, &Elements[i].e_mac, s
izeof(MACaddr_t)) == -1)
                {
                macs++;
               if(find_peb_mac(CASmacs[k].e_mac) == NULL)
               {
                /* Somebody in the NOC has deleted Ethernet */
                  /* address of this element... */
                  not_found++;
                  CASmacs[k].in_use = 'N';
               )
                )
         )
         if(macs == not_found)
         (
                /* There are no ethernet addresses for the element */
            del_peb_element(i);
                )

         )
      SaveConfig();
         )
   return ret;
}

/**************************************************************************
*
*       parse_gup(BYTE *buf, WORD len)
*
*       Description:
*                       Parse the GUP packet. Add new entries to the PACAU buffe
r.
*
*       Input:
*               buf                             - pointer to the message receive
d
*               len                             - length of the message received
*
*       Output:
*               Nothing
*
*       Returns:
*               0 if successfully parsed
*
**************************************************************************/

static parse_gup(BYTE *buf, WORD len)
{
        GUPid_t *p_gup_element;
   GUPhead_t        *p_gup_head;
   MUXpacau_t       *p_pacau_tmp;
   int i, curr_len = 0;
```

```
        int index, start, end;

   p_gup_head = (GUPhead_t *)buf;

   index = CCmpB(&p_gup_head->adapterstart, SerialNum, sizeof(ID));
        if (index == -1)
                start = 0;
        else
                start = p_gup_head->adapterstart.i[index] - SerialNum[i];

        index = CCmpB(&p_gup_head->adapterend, SerialNum, sizeof(ID));
        if (index == -1)
                end = 0;
        else
                end = p_gup_head->adapterend.i[index] - SerialNum[i];


   if(start <= 0 && end >= 0)
   {
        CDBVersion++;
      for(i = 0; i < p_gup_head->entries && curr_len < len; i++, len += GUP_LEN)
      {
        p_gup_element = (GUPid_t *)(buf + GUP_HEAD_LEN + i*GUP_LEN);
         if(CCmpB(&p_gup_element->adapternum, SerialNum, sizeof(ID)) == -1)
         {
                p_pacau_tmp = (MUXpacau_t *)(CASDBpacau.p_buffer + CASDBpacau.le
ngth);
            CASDBpacau.length += PACAU_LEN;
            CASDBpacau.entries++;
            CMovB(&p_gup_head->groupid, &p_pacau_tmp->groupid, sizeof(ID));
            p_pacau_tmp->version = p_gup_head->g_ver;
            CMovB(&p_gup_element->g_key, &p_pacau_tmp->g_key, sizeof(chunk));
            break;
                )
        )
        }
        return 0;
}

/**************************************************************************
*
*       update_peb(BYTE *buf, WORD len)
*
*       Description:
*                       Parse the UPDATE_PEB packet. Replace the old entry by th
e
*
*                       new one according to the Element ID.
*
*       Input:
*               buf                             - pointer to the message receive
d
*               len                             - length of the message received
*
*       Output:
*               Nothing
*
*       Returns:
*               0 if successfully parsed
*
**************************************************************************/

static update_peb(BYTE *buf, WORD len)
{
        int m;
   MUXpacau_t *pacau = NULL;
   MUXdacau_t *dacau = NULL;
```

```
    MUXpeb_t *old_peb, *new_peb, *found;
    int found_element;
    short ret_code = 0;
    unsigned long curr_version;

    if(len != 2 * PEB_LEN + PEB_HEAD_LEN)
         return(0);
         curr_version =
                 buf[0] +
          buf[1] * 256UL +
          buf[2] * 256UL * 256UL +
          buf[3] * 256UL * 256UL * 256UL;

         if(curr_version == CASDBpeb.version)
         return 0;
         CDBVersion++;
    CDBEthVersion++;
    CASDBpeb.version = curr_version;

    old_peb = (MUXpeb_t *)(buf + PEB_HEAD_LEN);
    new_peb = (MUXpeb_t *)(buf + PEB_HEAD_LEN + PEB_LEN);
    if((found = find_peb(old_peb->elementid, old_peb->version)) == NULL)
    /* Nothing to replace - ERROR */
         return(0);
         for(m = 0; m < MAXELEMENTS; m++)
         {
         if(UpdatedElements[m].in_use == 'Y')
         {
         /* We have received next replace element command,
           * it means, that the previous element
           * has already been updated at the
           * DataFeed Lan Gateway and
           * we can delete old address and keys
           * from adapter
           */

                         ret_code = DIODeleteAddress(UpdatedElements[m].channel,
                                 (BYTE *)&UpdatedElements[m].e_mac);

                         UpdatedElements[m].in_use = 'N';
                 }
         }
         /* Has been the element loaded before? */
    found_element = find_element_id(old_peb->elementid, old_peb->version);
    if(found_element != -1)
    {
         /* Yes. We are receiving this element now.
       * Let's keep old element's address
       * in the UpdatedElement table
       */
       CMovB(&Elements[found_element], &UpdatedElements[found_element],
                 sizeof(CDBelement_t));
                 /*
       * Trying to find a group for the element
       */
       pacau = find_pacau(new_peb->groupid, new_peb->grversion);
       dacau = find_dacau(new_peb->groupid, new_peb->grversion);
       if(dacau != NULL)
         pacau = (MUXpacau_t *)dacau;
                 if(pacau != NULL)
                 {
         /* Put element in the adapter
          * After this operation in the adapter will be
          * both old and new element's addresses and keys
          */
```

```
                         delay(120);
                         ret_code = DIOAddGroupAddress(Elements[found_element].ch
annel,
                                         (BYTE *)&Elements[found_element].e_mac,
(BYTE *)&pacau->g_key);
                 }
       else
       /* No group, Delete element */
         del_peb_element(found_element);
         }
    /* Change PEB database for this element */
    if(!ret_code)
    {
         replace_peb_element(found_element, new_peb->version);
       CMovB( (unsigned char *)new_peb, (unsigned char *)found, PEB_LEN - 2);
         }
         return(ret_code);
}

/**************************************************************************
*
*       AccessReceive(char *message)
*
*       Description:
*                       This routine checks to see if we've received any
*                       packets from the MLID. If we have, the data is copied
*                       from the ECB to the message and the ECB is returned to t
he
*                       LSL.
*
*
*       Input:
*               message                                                 - pointe
r to where to copy data
*
*       Output:
*               message and lroinfo filled in if successful
*
*       Returns:
*               0       .       if a packet has been received
*
***************************************************************************/

LONG    AccessReceive(char *message)
{
        ECB *ecb;

        /* Extract an ECB from the linked list if one exists */
        Disable();
        ecb = AccessECBHead;
        if (!ecb)
        {
                /* No ecb. Just return */
                Enable();
                return(-1);
        }
        AccessECBHead = ecb->ECB_NextLink;
        if (AccessECBHead == 0)
                AccessECBTail = 0;

        Enable();

        /* copy the data past the lroinfo to the message */
        CMovB(ecb->ECB_Fragment[0].FragmentAddress, message, ecb->ECB_Fragment[0
].FragmentLength);

        /* return the ecb to the LSL */
        CLSLReturnRcvECB(ecb);
#ifdef LOG_ECB_ACTIVITY
        FastLogMsg(LogECBHandle, (LogClientHandle, LogECBHandle, TRUE,
                                 "ACCESS return(%081x)\n", ecb));
#endif /* LOG_ECB_ACTIVITY */
        return(0);
)

/*****************************************************************************
 *
 *      AccessAdd(BYTE *message)
 *
 *      Description:
 *                      This routine is called when a packet is received
 *                      and is responsible for dispatching it.
 *
 *      Input:
 *              message                                 - packet data
 *              lroinfo                                 - Lro information
 *
 *      Output:
 *              Nothing
 *
 *      Returns:
 *              Nothing
 *
 *****************************************************************************/

int     AccessAdd(BYTE *message) (
  IndPacket_t *p_packet;
  int packet_type;
  int status;

  p_packet = (IndPacket_t *)message;

  if(p_packet->address[3] == 0x01)        /* CAS Individual address */
    switch(p_packet->payload[0]) (
    case SG_PACAU:
      packet_type = PACAU;
      break;
    case SG_ECAU:
      packet_type = ECAU;
      break;
    )
  else if(p_packet->address[0] == 0x03) (       /* Bypass key */
    switch(p_packet->payload[0]) (
    case PEB_PACKET:
      packet_type = PEB;
      break;
    case GUP_PACKET:
      packet_type = GUP;
      break;
    case PEB_UPDATE_PACKET:
      packet_type = PEB_UPDATE;
      break;
    default:
      packet_type = UNKNOWN;
      break;
    )
  )
  else
    packet_type = UNKNOWN;

  switch(packet_type) (
  case UNKNOWN:
    break;
```

```
  case PACAU:
    status = parse_pacau(p_packet->payload+1, p_packet->length-1);
    break;
  case ECAU:
    status = parse_ecau(p_packet->payload+1, p_packet->length-1);
    break;
  case PEB:
    status = parse_peb(p_packet->payload+1, p_packet->length-1);
    break;
  case GUP:
    status = parse_gup(p_packet->payload+1, p_packet->length-1);
    break;
  case PEB_UPDATE:
    status = update_peb(p_packet->payload+1, p_packet->length-1);
    break;
  )
  return(status);
)

/*****************************************************************************
 *
 *      parse_dacau(BYTE *buf,
 *                      int len)
 *
 *      Description:
 *                      Parse the new DACAU from the message received by the mod
em.
 *                      Replace the old DACAU buffer by the new one if the the v
ersion
 *                      matches what we have in the CASDBdacau version.
 *
 *      Input:
 *              buf                                 - pointer to the message receive
d
 *              len                                 - length of the message received
 *
 *      Output:
 *              Nothing
 *
 *      Returns:
 *              0 if successful
 *
 *****************************************************************************/

static int parse_dacau(BYTE *buf, int len)
(
   unsigned long curr_d_version;
   int ret = 0;

   curr_d_version =
     buf[0] +
     buf[1] * 256UL +
     buf[2] * 256UL * 256UL +
     buf[3] * 256UL * 256UL * 256UL;

   if(curr_d_version == CASDBdacau.version) (
     CDBVersion++;
     CASDBdacau.version = curr_d_version;
     CMovB(buf + DACAU_HEAD_LEN, CASDBdacau.p_buffer, len - DACAU_HEAD_LEN);
     CASDBdacau.length = len - DACAU_HEAD_LEN;
     CASDBdacau.entries = CASDBdacau.length / CASDBdacau.entry_len;
     check_df_groups();
   )
   else (
     ret = -1;
   )
```

```
        return ret;
}

/***********************************************************************
 *
 *      DacauResponse(BYTE *pdata,
 *                           int len,
 *                           timeoutFlag)
 *
 *      Description:
 *                      This is the callback routine pass in to DloScheduleRecei
ve()
 *                      which receives the response from the modem. If a timeout
 *                      hadn't occured, parse the message for the dacau info.
 *
 *      Input:
 *              pdata                                   - Pointer to response me
ssage
 *              len                                     - length of the message
received
 *              timeoutFlag                     - -non-zero if we timed out
 *
 *      Output:
 *              Nothing
 *
 *      Returns:
 *              Nothing
 *
***********************************************************************/

static void DacauResponse(BYTE *pdata, int len, int timeoutFlag)
{
        if (timeoutFlag)
        {
                DacauFlag = 1;
                UpdateFileStatus(MSG("Retry: Obtaining Encryption Keys", 258));
                return;
        }
//      EnterDebugger();
        if (parse_dacau(pdata, len) == 0)
        {
                WaitingForKeys = FALSE;
                CDBVersion++;
      SaveConfig();
                UpdateFileStatus(MSG("IDLE", 211));
        }
        else
        {
                DacauFlag = 1;
                UpdateFileStatus(MSG("Retry: Obtaining Encryption Keys", 259));
        }
}

/***********************************************************************
 *
 *      GetDacau(void)
 *
 *      Description:
 *                      Initiate the reception of the encryption keys for this a
dapter
 *                      by constructing a request, sending it to the modem, and
 *                      scheduling a receive routine to receive the response whi
ch
 *                      hopefully contains our encryption key.
 *
 *      Input:
```

```
 *              Nothing
 *
 *      Output:
 *              Nothing
 *
 *      Returns:
 *              Nothing
 *
***********************************************************************/

static void GetDacau(void)
{
        int k;
        BYTE c;

        memcpy(&DacauRequest.adapternum, SerialNumPacked, sizeof(ID));

        if( DIOSignText( (BYTE *)&DacauRequest, sizeof (DACAUrequest_t) - 8, Dac
auRequest.sign) != 0)
                return;

        for(k = 0; k < 8; k+=2)
        {
        c = DacauRequest.sign[k];
        DacauRequest.sign[k] = DacauRequest.sign[k+1];
      DacauRequest.sign[k+1] = c;
        }
        SlipSend((BYTE *)&DacauRequest, sizeof(DACAUrequest_t), 1, DLO_GETKEYS_T
IMEOUT);
        if (DloScheduleReceive(DacauResponse, 60, DACAU_RECEIVE) == 0)
                DacauFlag = 0;
}


/***********************************************************************
 *
 *      AccessMain(void *parm)
 *
 *      Description:
 *                      Main access thread which handles conditional access
 *                      packet reception. We'll start out by reading DPC.CFG.
 *                      If the serial number on the adapter is different from th
at
 *                      in DPC.CFG, we'll call out immediately to get the new ke
y.
 *                      Then we'll just sleep until there is a packet to [illegible]e.
 *
 *      Input:
 *              parm                                                    - ignored
 *
 *      Output:
 *              Nothing
 *
 *      Returns:
 *              Nothing
 *
***********************************************************************/

void    AccessMain(void *parm)
{
        LONG sn, ccode;
//      ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 212)};
//      ChannelConfig   channelCfg;
        BYTE address[8];
        BYTE x;
        LONG removedCount;
        parm = parm;

        /*
         * When MLID has been found, Open the conditional access channel.
         */

RegisterWithDriver:

        while(!ExitingFlag)
        {
                AccessAsleep = TRUE;
                delay(500);
                AccessAsleep = FALSE;

                if (DIOBoard != 0)
                {
                        removedCount = DIORemovedCount;
                        if (AccessChannel != -1 ||
                                        DIOOpenChannel(AccessAddress, AccessESR,
 &AccessChannel) == 0)
                        {
                                DIOGetSN(SerialNum);
                                sn = atol(SerialNum);
        NWsprintf(SerialNum, MSG("%061X", 213), sn);

                                pack_mac_addr(SerialNumPacked, 3, SerialNum, 6);
                        x = SerialNumPacked[0];
                        SerialNumPacked[0] = SerialNumPacked[2];
                        SerialNumPacked[2] = x;


                                MACbuildAddr(SerialNum, MAC_CAS_IND, 0, (MACaddr
_t *)address);
                                if (DIOAddAddress(AccessChannel, address) == 0)

//                              channelCfg.CfgChannel = AccessChannel;
//                              channelCfg.CfgNumAddresses = 1;
//                              MACbuildAddr(SerialNum, MAC_CAS_IND, 0, (MACaddr
_t *)channelCfg.CfgAddress);

//                              ecb.ECB_StackID = MLID_ADD_ADDRESS;
//                              ecb.ECB_Fragment[0].FragmentAddress = &channelCf
g;
//                              if (IoctlMlid(FDBboard, &ecb, FDBControlEntry) =
= 0)
                                        break;
                        }
                }
        }

        if (ExitingFlag)
        {
                DPCAccessPID = 0;
                return;
        }

        /*
        * Now lets open up the the DPC.CFG file.
        */

        ReadConfig();

        while(!ExitingFlag)
        {
```

```
                ccode = AccessReceive(AccessMessage);
                if (ccode)
                {
                        AccessAsleep = TRUE;
                        SuspendThread(DPCAccessPID);
                        AccessAsleep = FALSE;
                        if (removedCount != DIORemovedCount)
                                goto RegisterWithDriver;
                        continue;
                }

                AccessAdd(AccessMessage);
                if ( DacauFlag )
                        GetDacau();
        }

        DIOCloseChannel(AccessChannel);
        DPCAccessPID = 0;
}
```

```
#include "dpcagent.h"           /* Our header file */

LONG    DIOBoard = 0;
LONG    DIORemovedCount = 0;
LONG  DIOControlEntry = 0;
struct DriverStatsStructure* DIOStats = 0;

void DIORemove(void)
{
        int     i;

/*
 * Invalidate the DriverIO board. Increment removed count so that
 * other threads can detect that the driver has changed, even if
 * DIOBoard gets filled in. The other threads can than re-register
 * with the new driver.
 */

        DIOStats = 0;
        DIOBoard = 0;
        DIORemovedCount++;

/*
 * Force the NWSNUT menus to exit so that DPCAGENT can go back
 * to searching for a new adapter before allowing user to choose
 * options.
 */

        for(i = 0; i < 4; i++)
                NWSUngetKey(UGK_ESCAPE_KEY, UGK_NORMAL_KEY, NUTHandle);

/*
 * Force sleeping threads to wake up so that they can detect that
 * the adapter has been unloaded(DIORemovedCount will be different
 * from their local copy of the last removed count). The threads
 * should then spin(sleep) periodically until a new adapter is
 * present, and then re-register with the adapter if they need to.
 */

        if (AccessAsleep)
                ResumeThread(DPCAccessPID);

        if (InetAsleep)
                ResumeThread(DPCInetPID);

}

LONG DIORegisterWithAdapter(char *shortName)
{
        LONG    board;

        for(board = 0; board < NumberOfLANs; board++)
        {
                void (*ControlEntryPoint) (void)= NULL;
                if (CLSLGetMLIDControlEntry(board,
                                        &ControlEntryPoint) == ESUCCESS)
                {
                        struct DriverConfigurationStructure* config = 0;
                        if ((config = (struct DriverConfigurationStructure *) Co
mmandMlid(board, 0, (LONG)ControlEntryPoint)))
                        {
                                if (!CStrCmp(config->DShortName, shortName))
                                {
                                        ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC",
505)};
```

```
                                        void (*removeRoutine) () = DIORemove;

                                        ecb.ECB_StackID = MLID_REGISTER_AGENT;
                                        ecb.ECB_Fragment[0].FragmentAddress = &r
emoveRoutine;

                                        /* Call MLID */
                                        IoctlMlid(board, &ecb, (LONG)ControlEntr
yPoint);

                                        DIOBoard = board;
                                        DIOControlEntry = (LONG)ControlEntryPoin
t;

                                        return(0);
                                }
                        }
                }
        }
        return(-1);
}

void DIODeRegisterAgent(void)
{
        ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 506)};
        void (*nullRoutine) () = 0;

        if (DIOBoard)
        {
                ecb.ECB_StackID = MLID_REGISTER_AGENT;
                ecb.ECB_Fragment[0].FragmentAddress = &nullRoutine;

                /* Call MLID */
                IoctlMlid(DIOBoard, &ecb, DIOControlEntry);
                DIOBoard = 0;
        }
}


/***********************************************************************
*
*       DIOGetSN(char *serialNum)
*
*       Description:
*               This routine Gets the serial number from the adapter.
*               It is used for explicit requests of packages that are
*               for sale.
*
*       Input:
*               serialNum                       - Pointer to where to store seri
al number
*
*       Output:
*               serialNum                       - filled out if successful
*
*       Returns:
*               0 if successful
*
***********************************************************************/

LONG DIOGetSN(char *serialNum)
{
        LONG ccode;
        ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 108)};

        if (!DIOBoard)
                return(-1);
        ecb.ECB_StackID = MLID_GET_SN;
        ecb.ECB_Fragment[0].FragmentAddress = serialNum;

        /* Call MLID */
        ccode = IoctlMlid(DIOBoard, &ecb, DIOControlEntry);

        return(ccode);
}


/***************************************************************************
*
*       DIOSignText(char *textToSign,
*                   LONG textLength,
*                   char *signature)
*
*       Description:
*               This routine uses the adapter to calculate a signature
*               for the given text.
*               It is used for explicit requests of packages that are
*               for sale.
*
*       Input:
*               textToSign                      - string to be signed
*               textLength                      - length of string to be signed
*               signature                       - string to store signature
*
*       Output:
*               signature                       - filled out if successful
*
*       Returns:
*               0 if successful
*
****************************************************************************/

LONG DIOSignText(char *textToSign,
                LONG textLength,
                char *signature)
{
        LONG ccode;
        LONG fragment[3];
        ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 139)};

        if (!DIOBoard)
                return(-1);

        fragment[0] = (LONG)textToSign;
        fragment[1] = textLength;
        fragment[2] = (LONG)signature;

        ecb.ECB_StackID = MLID_SIGN_TEXT;
        ecb.ECB_Fragment[0].FragmentAddress = fragment;

        /* Call MLID */
        ccode = IoctlMlid(DIOBoard, &ecb, DIOControlEntry);

        return(ccode);
}

/***************************************************************************
*
*       EXPORTED FUNCTION
*
*       DPCGetMLIDStats(struct DriverStatsStructure **stats)
*
```

```
*       Description:
*               This routine fills in a pointer to the MLID stats
*               table.
*
*       Input:
*               stats                           - Pointer to where to store poin
ter to stats table
*
*       Output:
*               stats filled in if successful
*
*       Returns:
*               0               if MLID is active
*
****************************************************************************/

int DPCGetMLIDStats(struct DriverStatsStructure **stats)
{
        if (!DIOBoard)
                return(-1);

        *stats = DIOStats = (struct DriverStatsStructure *)
                CommandMlid(DIOBoard, 1, DIOControlEntry);

        return(0);
}


int DIOGetMLIDConfig(struct DriverConfigurationStructure **config)
{
        if (!DIOBoard)
                return(-1);

        *config = (struct DriverConfigurationStructure *)
                        CommandMlid(DIOBoard, 0, DIOControlEntry);

        return(0);
}

/***************************************************************************
*
*       DIOOpenChannel(BYTE *address, int (*esr)(), LONG *channel)
*
*       Description:
*                       This routine attempts to open an adapter channel
*                       the passed in bypass address, such as 0f 00 00 0[illegible]00.
*
*       Input:
*               address                                                 - Bypass
 address for open
*               esr
- ESR address for this channel
*               channel                                                 - where
channel number is returned
*
*       Output:
*               channel is filled out if successful
*
*       Returns:
*               0               if channel was opended
*
****************************************************************************/

LONG    DIOOpenChannel(BYTE *address, int (*esr)(), LONG *channel)
{
        ChannelConfig   cfg;
        LONG                    ccode;
        ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 142)};

        if (!DIOBoard) {
          if (DIORegisterWithAdapter(DPCName))
            return(-1);
        }

        /* Initialize channel to 0. MLID will overwrite this */
        cfg.CfgChannel = 0;

        /* Point ESR to our packet handler */
        cfg.CfgESR = esr;

        /* Number of addresses to add */
        cfg.CfgNumAddresses = 1;

        /* Address of 0f 00 00 00 00 00 */
        CMovB(address, cfg.CfgAddress, 6);

        ecb.ECB_StackID = MLID_OPEN_CHANNEL;
        ecb.ECB_Fragment[0].FragmentAddress = &cfg;

        /* Call MLID */
        ccode = IoctlMlid(DIOBoard, &ecb, DIOControlEntry);
        if (ccode == 0)
        {
                /* Store channel for close channel, add addr and delete addr */
                *channel = cfg.CfgChannel;
        }
        return(ccode);
}

/*************************************************************************
*
*       DIOCloseChannel(LONG channel)
*
*       Description:
*                       This routine closes a previously opened channel.
*
*       Input:
*               channel                                                 - channe
l to close
*
*       Output:
*               nothing
*
*       Returns:
*               0               if channel was closed
*
*************************************************************************/

LONG    DIOCloseChannel(LONG channel)
{
        LONG    ccode;
        ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 157)};

        if (!DIOBoard)
                return(-1);

        ecb.ECB_StackID = MLID_CLOSE_CHANNEL;
        ecb.ECB_Fragment[0].FragmentAddress = &channel;
        ccode = IoctlMlid(DIOBoard, &ecb, DIOControlEntry);

        return(ccode);
}
```

```
/*************************************************************************
*
*       DIODeleteAddress(BYTE *address)
*
*       Description:
*                       This routine deletes the filter address from the MLID.
*
*       Input:
*               address                                                 - address to del
ete
*
*       Output:
*               Nothing
*
*       Returns:
*               0 if successful
*
*************************************************************************/

LONG    DIODeleteAddress(LONG channel, BYTE *address)
{
        ChannelConfig   channelCfg;
        LONG ccode;
        ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 180)};

        if (!DIOBoard)
                return(-1);

        channelCfg.CfgChannel = channel;
        channelCfg.CfgNumAddresses = 1;
        CMovB(address, channelCfg.CfgAddress, 6);

        ecb.ECB_StackID = MLID_DEL_ADDRESS;
        ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
        ccode = IoctlMlid(DIOBoard, &ecb, DIOControlEntry);
        return(ccode);

}

LONG DIOAddAddress(LONG channel, BYTE *address)
{
        ChannelConfig   channelCfg;
        LONG ccode;
        ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 207)};

        if (!DIOBoard)
                return(-1);

        channelCfg.CfgChannel = channel;
        channelCfg.CfgNumAddresses = 1;
        CMovB(address, channelCfg.CfgAddress, 8);

        ecb.ECB_StackID = MLID_ADD_ADDRESS;
        ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
        ccode = IoctlMlid(DIOBoard, &ecb, DIOControlEntry);
        return(ccode);

}

LONG DIOAddGroupAddress(LONG channel, BYTE *address, BYTE *groupAddress)
{
        ChannelConfig   channelCfg;
        LONG ccode;
        ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 209)};
        BYTE key[8];
        if (!DIOBoard)
                return(-1);

        channelCfg.CfgChannel = channel;
        channelCfg.CfgNumAddresses = 1;
        CMovB(address, channelCfg.CfgAddress, 8);
        CMovB(groupAddress, key, 8);
        reverse_key((BYTE*)&key);
        CMovB(key, channelCfg.CfgGroupKey, 8);
        CMovB(&MagicKey, channelCfg.CfgElementKey, 8);

        ecb.ECB_StackID = MLID_ADD_ADDRESS;
        ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
        ccode = IoctlMlid(DIOBoard, &ecb, DIOControlEntry);
        return(ccode);

}

int DIOAddHIAddr(unsigned char channel, BYTE *hiAddr)
{
        chunk key;
  ID hi_id;
  int i, ret = CAS_ERROR;
        ChannelConfig   channelCfg;
        ECB ecb = (0, 0, 0, 0, 0, MSG("DRCTPC", 471));

        if (!DIOBoard)
                return(-1);

  for(i = 0; i < 3; i++)
        hi_id.i[i] = 0x00;
        make_hi_key(&key);
  channelCfg.CfgChannel = channel;
  channelCfg.CfgNumAddresses = 1;

        CMovB(hiAddr, channelCfg.CfgAddress, 8);
/* Some strange things ... */
        reverse_key((BYTE *)&key);
  CMovB(&key, channelCfg.CfgGroupKey, 8);
  CMovB(&key, channelCfg.CfgElementKey, 8);

//              channelCfg.CfgChannel = FDBChannel;
//              channelCfg.CfgESR = FDB_ESR;
//              channelCfg.CfgNumAddresses = 1;
//              CMovB(&addr, channelCfg.CfgAddress, 8);
//              CMovB(&pacau->g_key, key, 8);
//              reverse_key(&key);
//              CMovB(key, channelCfg.CfgGroupKey, 8);
//              CMovB(&MagicKey, channelCfg.CfgElementKey, 8);
//
        channelCfg.CfgESR = (int (*)())0xffffffff;
        ecb.ECB_StackID = MLID_ADD_ADDRESS;
        ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
//      CMovB(&addr, node->file_address, 6);

        ret = IoctlMlid(DIOBoard, &ecb, DIOControlEntry);

//  if((ret = WBicddAddAddress
//              ((BICDD_CHANNEL_CONFIG FAR *)&channel_config))!=CAS_OK)

  return ret;
}

int     DIORegisterSend(int (*sendRoutine)(TCB *))
{
```

```
        ECB ecb = (0, 0, 0, 0, 0, MSG("DRCTPC", 481));

        if (!DIOControlEntry)
                return(-1);

        ecb.ECB_StackID = MLID_REGISTER_SEND_ROUTINE;
        ecb.ECB_Fragment[0].FragmentAddress = &sendRoutine;

        /* Call MLID */
        IoctlMlid(DIOBoard, &ecb, DIOControlEntry);
        return(0);
}

int     DIOObtainReturnTCB(void (**returnTCBRoutine)(TCB *))
{
        ECB ecb = (0, 0, 0, 0, 0, MSG("DRCTPC", 147));

        if (!DIOControlEntry)
                return(-1);

        ecb.ECB_StackID = MLID_RETURN_TCB_ROUTINE;
        ecb.ECB_Fragment[0].FragmentAddress = returnTCBRoutine;

        /* Call MLID */
        IoctlMlid(DIOBoard, &ecb, DIOControlEntry);
        return(0);
}


int DPCGetIPAddress(LONG* ip) (
  LONG id;
  LONG (*ctl)(LONG board, ...);
  char buf[80];
  char* s = buf;

  if (CLSLGetStackIDFromName("\002IP", &id))
    return 1;
  if (CLSLGetProtocolControlEntry(id, DIOBoard, &ctl))
    return 2;
  if (GetProtocolStringForBoard(ctl, DIOBoard, buf))
    return 3;
  s = strpbrk(buf, "0123456789");
  if (!s)
    return 4;
  *ip = inet_addr(s);
  if (*ip == (LONG)(-1))
    return 5;
  return 0;
}
```

```
#include "dpcagent.h"
#include <string.h>
#include <stdlib.h>

LONG DPCMaxConnections = 3;
int PackageDelivery = FALSE;

DloCfg_t DloCfg = {
  1330,                                 // freq
  (198 << 24) | (77 << 16) | (117 << 8) | 21, // ip_address
  (198 << 24) | (77 << 16) | (117 << 8) | 66, // gateway_address
  1500,                                 // mtu
  MSG("ATDT1-800-332-8071", 100),       // tinet_phone_num
  MSG("ATDT1-800-825-3954", 187),       // pdeliv_phone_num
  "",                                   // dialout_prefix
  300,                                  // tinet_inactivity_timer
  2,                                    // pdeliv_inactivity_timer
  1,                                    // modem_type
  120,                                  // packet_lifetime - Max time to keep da
ta in buffer
  60,                                   // call_setup_timeout
  MSG("ATH0", 189),                     // hangup_str
  MSG("NO CARRIER", 190),               // disconnect_str
  MSG("+++", 191),                      // escape_str
  MSG("CONNECT", 192),                  // connect_str
  MSG("ATE1Q0V1X4&C1&D2 S7=60 S11=55", 193), // init_str
  1024,                                 // max_db_entries
  5,                                    // tinet_baud_index(19200)
  0,                                    // pdeliv_baud_index(2400)
  1024 * 8,                             // async_buffer_size(8K)
  FALSE,                                // auto_login
  "",   "", "", "", "", "", "", "", "", // wait_for_1..wait_for_9
  "",   "", "", "", "", "", "", "", "", // send_1..send_9
  OUT_SLIP,                             // out_protocol
  TRUE,                                 // obsolete(was tunnel)
  10, 5, 5, 5, 5, 5, 5, 5, 5,           // wait_timeout_1..wait_timeout_9
  "",                                   // ppp_login .
  "",                                   // ppp_password
  1400,                                 // ppp_mru
  0,                                    // ppp_accm
  "00000000",                           // base_license
  "",                                   // key
  -1,                                   // net_interface
  "\0\0\0\0\0\0",                       // net_addr
  { "", "", "", "", "", "", "", "", "" }, // add_license
};

/*******************************************************************
 *
 * EXPORTED FUNCTION
 *
 *      DPCUpdateConfig(void)
 *
 *      Description:
 *              This routine opens \DIRECPPC\DB\MODEM.CFG and updates our global
 
 *              structures. If the file doesn't exist, or if its an older versio
n
 *              (because its smaller), read in what you can and write out a new
one.
 *
 *      Input:
 *
 *      Output:
 *
```

```
 *      Returns:
 *              0 if successful
 *              -1 unable to open or create file
 *              -2 unable to update file
 *
 ********************************************************************/

void ConfigSanityCheck(int handle)
{
        int changeFlag = 0;
        int i;
        LONG *timeout;

        if (DloCfg.wait_timeout_1 < 2 || DloCfg.wait_timeout_1 > 60)
        {
                changeFlag++;
                DloCfg.wait_timeout_1 = 10;
        }

        for (i = 0, timeout = &DloCfg.wait_timeout_2; i < 8; i++, timeout++)
        {
                if (*timeout < 2 || *timeout > 60)
                {
                        changeFlag++;
                        *timeout = 5;
                }
        }

        if (changeFlag)
        {
                lseek(handle, 0, SEEK_SET);
                write(handle, &DloCfg, sizeof(DloCfg));
        }

}

int DPCUpdateConfig(void)
{
        int     ccode = -1;
        int handle;

        handle = open(MSG("SYS:DIRECPC\\DB\\MODEM.CFG", 194),
                      O_RDWR | O_CREAT,
                      S_IWRITE | S_IREAD);
        if (handle != -1)
        {
                if ( read(handle, &DloCfg, sizeof(DloCfg)) != sizeof(DloCfg))
                {
                        lseek(handle, 0, SEEK_SET);
                        if (write(handle, &DloCfg, sizeof(DloCfg)) != sizeof(Dlo
Cfg))
                        {
                                ccode = -2;
                        }
                }
                ConfigSanityCheck(handle);
                close(handle);
                ccode = 0;
        }
        DloCfg.ip_address = htonl(DloCfg.ip_address);
        DloCfg.gateway_address = htonl(DloCfg.gateway_address);

        return(ccode);
}

void DPCUpdateConfigFile(void) {
        int handle;

        DloCfg.ip_address = ntohl(DloCfg.ip_address);
        DloCfg.gateway_address = ntohl(DloCfg.gateway_address);
        handle = open(MSG("SYS:DIRECPC\\DB\\MODEM.CFG", 335),
                      O_RDWR | O_CREAT,
                      S_IWRITE | S_IREAD);
        if (handle != -1)
        {
                write(handle, &DloCfg, sizeof(DloCfg));
                close(handle);
        }
        DloCfg.ip_address = htonl(DloCfg.ip_address);
        DloCfg.gateway_address = htonl(DloCfg.gateway_address);
}


#define SerialWarn(s) sprintf(warnbuf, "\r\nDPCN: detected a bad serial number:
%8.8s\r\n", (char*)(s)), ConsolePrintf(warnbuf), RingTheBell()


static inline unsigned long find_and_clear_low_bit(unsigned long* val) {
  unsigned long low = *val;
  if (low == 0)
    return 0;
  *val &= low - 1;
  return (*val ^ low);
}


static inline int parity(LONG serial) {
  int i;
  for (i = 0; find_and_clear_low_bit(&serial); ++i)
    ;
  return (i & 1);
}


static inline int UserCount(BYTE* s) {
  LONG serial = 0;
  int shift;

  s += 3;
  for (shift = 16; shift >= 0; shift -= 4) {
    serial |= (*s - ((*s >= 'A') ? 56 : 0x30)) << shift;
    ++s;
  }
  return 5 * (((serial & 0x00002) >> 1) |
              ((serial & 0x20000) >> 16) |
              ((serial & 0x02000) >> 11) |
              ((serial & 0x00040) >> 3));
}


void DPCSetMaxConnections(LONG* sum) {
  char warnbuf[120];
  int users;
  int pd = 0;
  int i;

  sum[0] = sum[1] = (-1);

  /* re-read modem.cfg file */
  i = DloCfg.ip_address;
  DPCUpdateConfig();
  DloCfg.ip_address = i;
```

```
  if (strncmp(DloCfg.base_license, "Helius, Inc.", 8) == ESUCCESS) {
    DPCMaxConnections = 108;
    PackageDelivery = 1;
    memcpy(sum, DloCfg.base_license, 2 * sizeof(LONG));
    return;
  }

  /* check for old license info */
  if (atoi(DloCfg.base_license) < 02) {
    sprintf(warnbuf,
            "\r\nDPCN: deactivated old serial number: %8.8s\r\n",
            DloCfg.base_license);
    ConsolePrintf(warnbuf);
    RingTheBell();
    return;
  }

  /* handle base license first */
  memcpy(sum, DloCfg.base_license, 2 * sizeof(LONG));
  if (parity(sum[0]) == 0) {
    SerialWarn(DloCfg.base_license);
    return;
  }
  if (parity(sum[1]) == 0) {
    SerialWarn(DloCfg.base_license);
    return;
  }
  users = UserCount(DloCfg.base_license);
  if (DloCfg.base_license[2] == '*')
    pd = 1;

  /* now handle additive licenses */
  for (i = 0; i < 9; ++i) {
    int j;
    LONG serial[2];
    if (DloCfg.add_license[i][0] == 0)
      continue;
    /* check for duplicate license */
    for (j = i - 1; j >= 0; --j) {
      if (memcmp(DloCfg.add_license[i],
                 DloCfg.add_license[j],
                 sizeof(DloCfg.add_license[i])) == ESUCCESS) {
        memset(DloCfg.add_license[i], 0, sizeof(DloCfg.add_license[i]));
        DPCUpdateConfigFile();
        sprintf(warnbuf,
                "\r\nDPCN: deleted duplicate license number: %8.8s\r\n",
                DloCfg.add_license[j]);
        ConsolePrintf(warnbuf);
        RingTheBell();
        goto nextLicense;
      }
    }
    memcpy(serial, DloCfg.add_license[i], sizeof(serial));
    if (parity(serial[0]) == 1) {
      SerialWarn(DloCfg.add_license[i]);
      return;
    }
    if (parity(serial[1]) == 1) {
      SerialWarn(DloCfg.add_license[i]);
      return;
    }
    users += UserCount(DloCfg.add_license[i]);
    sum[0] += serial[0];
    sum[1] += serial[1];
    if (DloCfg.add_license[i][2] == '*')
```

```
      pd = 1;
  nextLicense:
    ;
  }

  DPCMaxConnections = users * 4;
  PackageDelivery = pd;
}
```

```
#include <dpcagent.h>           /* Our header file */

#define USE_AIO_DEADMAN 0
#define TRACE_STATE 0

static char* printables = MSG("A..Za..z0..9 -,'~!@#$%^&*()_+={}()|;:'.<>?\\/\"",
 523);
#define dial_chars printables
#define modem_control_chars printables

int     AIOPortHandle = -1;
LONG    AIOWriteBufferSize = 0;
int     AIOGlobalPort = 0;
static BYTE AIOBaudRateDefines[] =
(
        AIO_BAUD_2400,                          /* 0 */
        AIO_BAUD_3600,                          /* 1 */
        AIO_BAUD_4800,                          /* 2 */
        AIO_BAUD_7200,                          /* 3 */
        AIO_BAUD_9600,                          /* 4 */
        AIO_BAUD_19200,                 /* 5 */
        AIO_BAUD_38400,                 /* 6 */
        AIO_BAUD_57600,                 /* 7 */
        AIO_BAUD_115200                 /* 8 */

);

int     DloState = DLOS_IDLE;
LONG    DloTimer = 0;
LONG    DloPacketLifeTimer = 0;

LONG    DloConn = DLO_CONN_IDLE;
LONG    DloNextConn = DLO_CONN_IDLE;

LONG    DloPXmitCount = 0;
LONG    DloPMaxBufferSize = DLOBUFSIZE;
BYTE    DloPXmitBuffer[DLOBUFSIZE];
LONG    DloPInactivityTimer = 10 * 19;

LONG    DloIXmitCount = 0;
LONG    DloIMaxBufferSize = DLOBUFSIZE;
BYTE    DloIXmitBuffer[DLOBUFSIZE];
LONG    DloIInactivityTimer = 60 * 19;

LONG    DloLastKnownTickCount;
LONG    DloRcvCount = 0;
LONG    DloRcvIndex = 0;
LONG    DloReadIndex = 0;
BYTE    DloRcvBuffer[DLOBUFSIZE];
LONG    DloCommandIndex = 0;
BYTE    DloCommandBuffer[DLOCMDBUFSIZE];
LONG    DloLastDCD = 0;

char    ConnectBaudStr[DLOCMDBUFSIZE] = (0);

static char *DloCompareString[DLOENUM] =
(
        "",
        DloCfg.connect_str,
        DloCfg.disconnect_str,
        "",
        MSG("OK", 219),
        MSG("BUSY", 220),
        MSG("NO DIAL TONE", 221),
        MSG("RING", 222),
```

```
        MSG("NO ANSWER", 99)
);

/*
 *      Variables used by DloScheduleReceive().
 */

void (*DloCallBack)(BYTE *pdata, int len, int timeoutFlag) = 0;
int DloCallBackTimeout = 0;
int DloCallBackWait = 0;
BYTE DloCallBackBuffer[4000];
int DloCallBackIndex = 0;
int DloCallBackStarted = 0;
int DloCallBackEscape = 0;
int     DloCallBackType = 0;

/*
 * RS232.NLM Default Init String
 */
//BYTE                          ModemInitString[] = MSG("ATH0Q0V1X4S0", 
/*
 * Digitan DS144FVM, Multitech Auto Reliable and Practical
 * Peripherals V.34 Default Init String
*/
//BYTE                          ModemInitString[] = "ATE1Q0V1X4&C1&D2&K0 S7=60 S
11=55";
/*
 * Hayes-Compatible Modem Default Init String
 */
//BYTE                          ModemInitString[] = MSG("ATE1Q0V1X4&C1&D2 S7=60
S11=55", 156);
/*
 * Intel 144e Faxmodem and Motorola UDS V.3225
 * Default Init String
 */
//BYTE                          ModemInitString[] = "ATE1Q0V1X4&C1&D2\G S7=60 S1
1=55";
/*
 * U.S. Robotics Default Init String
 */
//BYTE                          ModemInitString[] = MSG("ATE1Q0V1X4&C1&D2&K0&H0&
I0 S7=60 S11=55", 100);


/* internal function prototypes */

void InitializeAIO(void);
void SendAIOData(BYTE *data, LONG length);
static void DloStartTimer( LONG ticks );
//void DloStopTimer( void );
static void DloStopPacketLifeTimer( void );
static void DloStartPacketLifeTimer( void );
static void StateMachine (int event);
static void WriteCommXmitBuffer( void );


void HexAsciiDump(const unsigned char* buffer, unsigned int len)
(
        char display[80];

        while (len >= 16)
        {
                NWprintf(MSG("%02x %02x %02x %02x %02x %02x %02x %02x   %02x %02
x %02x %02x %02x %02x %02x   %c%c%c%c%c%c%c%c  %c%c%c%c%c%c%c%c\n", 127),
                              buffer[0],
                              buffer[1],
                           buffer[2],
                           buffer[3],
                           buffer[4],
                           buffer[5],
                           buffer[6],
                           buffer[7],
                           buffer[8],
                           buffer[9],
                           buffer[10],
                           buffer[11],
                           buffer[12],
                           buffer[13],
                           buffer[14],
                           buffer[15],
                           isprint(buffer[0]) ? buffer[0] : '.',
                           isprint(buffer[1]) ? buffer[1] : '.',
                           isprint(buffer[2]) ? buffer[2] : '.',
                           isprint(buffer[3]) ? buffer[3] : '.',
                           isprint(buffer[4]) ? buffer[4] : '.',
                           isprint(buffer[5]) ? buffer[5] : '.',
                           isprint(buffer[6]) ? buffer[6] : '.',
                           isprint(buffer[7]) ? buffer[7] : '.',
                           isprint(buffer[8]) ? buffer[8] : '.',
                           isprint(buffer[9]) ? buffer[9] : '.',
                           isprint(buffer[10]) ? buffer[10] : '.',
                           isprint(buffer[11]) ? buffer[11] : '.',
                           isprint(buffer[12]) ? buffer[12] : '.',
                           isprint(buffer[13]) ? buffer[13] : '.',
                           isprint(buffer[14]) ? buffer[14] : '.',
                           isprint(buffer[15]) ? buffer[15] : '.');
                buffer += 16;
                len -= 16;
        }

        if (len)
        {
                /* the basic theory here is to build the buffer in place and the
n insert the extra spaces */
                unsigned int n = 0;
                register char* d = display;

                while (n < len)
                {
                        NWsprintf(d, MSG("%02x ", 483), buffer[n]);
                        d += 3;
                        display[(16 * 3 + 2) + n] = isprint(buffer[n]) ? buffer[
n] : '.';
                        ++n;
                }
                display[(16 * 3 + 2) + len] = 0;
                memset(d, ' ', (16 - len) * 3 + 2);
                d = display + (16 / 2 * 3);
                memmove(d + 2, d, sizeof(display) - (16 / 2 * 3) - 2);
                d[0] = d[1] = ' ';
                d = display + (16 * 3) + 4 + 8;
                memmove(d + 2, d, 9);
                d[0] = d[1] = ' ';
                puts(display);
        }
}

void DloUpdateModemStr( void )
{
        if (DloState == DLOS_IDLE)
                UpdateModemStr(MSG("Modem Status: IDLE
               \n", 201));
```

```
        else if (DloState == DLOS_INIT)
                UpdateModemStr(MSG("Modem Status: Initializing Modem
               \n", 240));
        else if (DloState == DLOS_DIAL)
                UpdateModemStr(MSG("Modem Status: Dialing
               \n", 236));
        else if (DloState == DLOS_REDL)
                UpdateModemStr(MSG("Modem Status: Redialing
               \n", 235));
        else if (DloState == DLOS_CONN)
        {
                if (DloConn == DLO_CONN_PACKAGE)
                        UpdateModemStr(MSG("Modem Status: Connected to Package D
elivery                \n", 241));
                else
                {
                        if (ConnectBaudStr[0])
                        {
                                BYTE connectStr[80];

                                NWsprintf(connectStr, MSG("Modem Status: Connect
ed to Internet at%.24s\n", 473), ConnectBaudStr);
                                UpdateModemStr(connectStr);
                        }
                        else
                        {
                                UpdateModemStr(MSG("Modem Status: Connected to I
nternet                        \n", 184));
                        }
                }
        }
        else
        {
                if (DloConn == DLO_CONN_PACKAGE)
                        UpdateModemStr(MSG("Modem Status: Disconnecting from Pac
kage Delivery          \n", 101));
                else
                        UpdateModemStr(MSG("Modem Status: Disconnecting from Int
ernet                  \n", 475));
        }
}

int DloGetWriteBufferSpace( void )
{
        LONG writeCount = 0;

        if (!AIOWriteBufferSize)
                return(2048);

        AIOGetPortStatus(AIOPortHandle, &writeCount, NULL, NULL, NULL, NULL, NUL
L);

        return(AIOWriteBufferSize - writeCount);

}

int DloAndCommEmpty (void)
{
        if (DloConn == DLO_CONN_PACKAGE)
                return(DloPXmitCount == 0);
        else
                return(DloIXmitCount == 0);
}

void DloFlushReceive(void)
{
        AIOFlushBuffers(AIOPortHandle, AIO_FLUSH_READ_BUFFER);
        DloRcvCount = 0;
        DloRcvIndex = 0;
        DloReadIndex = 0;
}

void DloStartConn(int timeout)
{
        if (DloNextConn == DLO_CONN_IDLE)
        {
                /* Assume package delivery connection first */
                DloNextConn = DLO_CONN_PACKAGE;

                if (timeout == DLO_PACKAGE_TIMEOUT)
                {
                        DloPInactivityTimer = DloCfg.pdeliv_inactivity_timer * 1
9;
                }
                else if (timeout == DLO_INET_TIMEOUT)
                {
                        DloIInactivityTimer = DloCfg.tinet_inactivity_timer * 19
;
                        DloNextConn = DLO_CONN_INET;
                }
                else if (timeout == DLO_GETKEYS_TIMEOUT)
                {
                        DloPInactivityTimer = 30 * 19;
                }
                else if (timeout > 2)
                {
                        DloPInactivityTimer = timeout * 19;
                }
                else
                {
                        DloPInactivityTimer = 2 * 19;
                }
        }

        if (DloState == DLOS_CONN && DloNextConn == DloConn)
        {
                DloNextConn = DLO_CONN_IDLE;
                StateMachine(DLOE_SEND);
                return;
        }

        if (DloConn == DLO_CONN_IDLE)
        {
                StateMachine (DLOE_SEND);
                DloConn = DloNextConn;
                DloNextConn = DLO_CONN_IDLE;
        }
        else if (DloConn == DLO_CONN_INET)
        {
                /* Package waiting for internet. Cause it to timeout quickly */
                DloStartTimer(19);
        }
}


int     BaudRateHandler(int option, void *parameter)
{
        parameter = parameter;
        return option;
}

static void NoSortHandler(LIST *head, LIST *tail, NUTInfo *handle)
{
```

```
        head = head;
        tail = tail;
        handle = handle;

        return;
}

void PDConfiguration(void)
{
        int     i;
        void (*oldSortFunction)(LIST *, LIST *, NUTInfo *);
        int     save;
        DloCfg_t        tmpDloCfg;
        MFCONTROL *mfctl1;
        int baud;

        CMovB(&DloCfg, &tmpDloCfg, sizeof(DloCfg_t));

        NWSInitForm(NUTHandle);

        NWSGetListSortFunction(NUTHandle, &oldSortFunction);
        NWSSetListSortFunction(NUTHandle, NoSortHandler);

        i = 0;
        NWSAppendCommentField(i, 2, MSG("Package Phone           : ", 311), NUTH
andle);
        NWSAppendStringField(i, 28, 30, NORMAL_FIELD, tmpDloCfg.pdeliv_phone_num
,
                                dial_chars, F_NO_HELP, NUTHandle);

        i++;
        baud = tmpDloCfg.pdeliv_baud_index;
        NWSAppendCommentField(i, 2, MSG("Package Baud            : ", 313), NUTH
andle);
        mfctl1 = NWSInitMenuField(InxMSG("Baud Rate", 314), 10, 40, BaudRateHand
ler, NUTHandle);
        NWSAppendToMenuField(mfctl1, InxMSG("2400", 315), 0, NUTHandle);
        NWSAppendToMenuField(mfctl1, InxMSG("3600", 316), 1, NUTHandle);
        NWSAppendToMenuField(mfctl1, InxMSG("4800", 317), 2, NUTHandle);
        NWSAppendToMenuField(mfctl1, InxMSG("7200", 318), 3, NUTHandle);
        NWSAppendToMenuField(mfctl1, InxMSG("9600", 319), 4, NUTHandle);
        NWSAppendToMenuField(mfctl1, InxMSG("19200", 320), 5, NUTHandle);
        NWSAppendToMenuField(mfctl1, InxMSG("38400", 321), 6, NUTHandle);
        NWSAppendToMenuField(mfctl1, InxMSG("57600", 322), 7, NUTHandle);
        NWSAppendToMenuField(mfctl1, InxMSG("115200", 323), 8, NUTHandle)
        NWSAppendMenuField(i, 28, NORMAL_FIELD, &baud, mfctl1, NULL, NUTH     ;

        i++;
        NWSAppendCommentField(i, 2, MSG("Package Inactivity(sec) : ", 324), NUTH
andle);
        NWSAppendIntegerField(i, 28, NORMAL_FIELD, (int *)&tmpDloCfg.pdeliv_inac
tivity_timer, 1, 65535, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Max Database Entries    : ", 327), NUTH
andle);
        NWSAppendIntegerField(i, 28, NORMAL_FIELD, (int *)&tmpDloCfg.max_db_entr
ies, 1, 65535, F_NO_HELP, NUTHandle);

        i++;

        save = NWSEditPortalForm(InxMSG("Package Delivery Configuration Editor",
 308),
                12, 40,
/* center line, column  */
                i, 76,
/* form height, width   */
                F_VERIFY, F_NO_HELP,    /* Control flags, help message */
                InxMSG("Save Changes?", 328),
                NUTHandle);                             /* Confirm message, hand
le */

        NWSSetListSortFunction(NUTHandle, oldSortFunction);
        NWSDestroyForm(NUTHandle);

        if (!save)
                return;

        tmpDloCfg.pdeliv_baud_index = baud;
        CMovB(&tmpDloCfg, &DloCfg, sizeof(DloCfg_t));

        DPCUpdateConfigFile();
}

LONG    ModifyPPPConfig(FIELD *fp, int key, int *changed, NUTInfo *handle)
{
        int     i;
        DloCfg_t        *tmpDloCfg;
        LONG save;

        key = key;
        changed = changed;
        handle = handle;

        tmpDloCfg = (DloCfg_t *)fp->customData;

        if (NWSPushList(NUTHandle) == 0)
                return K_SELECT;

        NWSInitForm(NUTHandle);

        i = 0;

        NWSAppendCommentField(i, 2, MSG("Authentication User Name: ", 516), NUTH
andle);
        NWSAppendStringField(i, 28, 30, NORMAL_FIELD, tmpDloCfg->ppp_login,
                        printables, F_NO_HELP, NUTHandle);

        i++;

        NWSAppendCommentField(i, 2, MSG("Authentication Password : ", 578), NUTH
andle);
        NWSAppendStringField(i, 28, 30, NORMAL_FIELD, tmpDloCfg->ppp_password,
                        printables, F_NO_HELP, NUTHandle);

        i+=2;
        NWSAppendCommentField(i, 2, MSG("Maximum Receive Unit    : ", 579), NUTH
andle);
        NWSAppendIntegerField(i, 28, NORMAL_FIELD, (int *)&tmpDloCfg->ppp_mru, 6
4, 16384, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Asynch. Control Char Map: 0x", 580), NU
THandle);
        NWSAppendHexField(i, 30, NORMAL_FIELD, (int *)&tmpDloCfg->ppp_accm, 0, 0
xffffffff, F_NO_HELP, NUTHandle);

        i++;
        save = NWSEditPortalForm(InxMSG("PPP Configuration Editor", 510),
                12, 40,
/* center line, column  */
                i, 78,
```

```
/* form height, width   */
                F_NOVERIFY, F_NO_HELP,  /* Control flags, help message */
                InxMSG("Save Changes?", 511),
                NUTHandle);                             /* Confirm message, hand
le */

//      if (save)
//      {
//              CMovB(tmpDloCfg->ppp_login, DloCfg.ppp_login, (30*2)+4+4 );
//              DPCUpdateConfigFile();
//      }
        NWSDestroyForm(NUTHandle);
        NWSPopList(NUTHandle);
        return K_SELECT;

}

LONG    ModifyNetConfig(FIELD* fp, int key, int* changed, NUTInfo* handle)
{
        int i;
        DloCfg_t* tmpDloCfg = (DloCfg_t*)fp->customData;
        LONG interface;
        char hw_addr[18];
        int save;

        key = key;                      /* not used */
        changed = changed;              /* not used */
        handle = handle;                /* not used */

        if (NWSPushList(NUTHandle) == 0)
                return K_SELECT;

        NWSInitForm(NUTHandle);

retry:
        i = 0;
        interface = tmpDloCfg->net_interface;
        NWSAppendCommentField(i, 2, "Interface: ", NUTHandle);
        NWSAppendUnsignedIntegerField(i, 35, NORMAL_FIELD,
                                &interface,
                                0, 256,
                                F_NO_HELP, NUTHandle);
        ++i;

        NWSAppendCommentField(i, 2, "Router Mac Address: ", NUTHandle);
        sprintf(hw_addr, "%02x-%02x-%02x-%02x-%02x-%02x",
                tmpDloCfg->net_addr[0],
                tmpDloCfg->net_addr[1],
                tmpDloCfg->net_addr[2],
                tmpDloCfg->net_addr[3],
                tmpDloCfg->net_addr[4],
                tmpDloCfg->net_addr[5]);
        NWSAppendStringField(i, 35, 17, NORMAL_FIELD,
                        hw_addr,
                        "0..9A..Fa..f-",
                        F_NO_HELP, NUTHandle);
        ++i;

        save = NWSEditPortalForm(InxMSG("Network Route Configuration Editor", 67
9),
                                12, 40,         /* center line & column */
                                i, 78, /* form height & width */
                                F_NOVERIFY, F_NO_HELP,
                                NULL,
                                NUTHandle);
        if (save)
```

```
	{
		LONG net_addr[6];
		void (*ControlEntryPoint)(void) = 0;
		struct DriverConfigurationStructure* dvrCfg = 0;
		if (sscanf(hw_addr, "%2x-%2x-%2x-%2x-%2x-%2x",
				&net_addr[0],
				&net_addr[1],
				&net_addr[2],
				&net_addr[3],
				&net_addr[4],
				&net_addr[5]) != 6)
		{
			NWSAlert(12, 40, NUTHandle,
				InxMSG("Format error in Mac Address", 680));
			goto retry;
		}
		for (i = 0; i < 6; ++i)
			tmpDloCfg->net_addr[i] = (BYTE)net_addr[i];
		if (CLSLGetMLIDControlEntry(interface,
						&ControlEntryPoint))
		{
			NWSAlert(12, 40, NUTHandle,
				InxMSG("Interface not found", 681));
			goto retry;
		}
		dvrCfg = (struct DriverConfigurationStructure *)
			CommandMlid(interface, 0, (LONG)ControlEntryPoint);
		if (!dvrCfg)
		{
			NWSAlert(12, 40, NUTHandle,
				InxMSG("Could not retrieve Interface Configurat
ion Table", 682));
			goto retry;
		}
		if ((dvrCfg->DModeFlags & (1 << 6)) == 0)
		{
			NWSAlert(12, 40, NUTHandle,
				InxMSG("Interface does not support \"Raw Send\"
", 683));
			goto retry;
		}
		if (dvrCfg->DMediaID != 2)
		{
			NWSAlert(12, 40, NUTHandle,
				InxMSG("Interface does not support ETHERNET_II
frames", 684));
			goto retry;
		}
		if (!CStrCmp(dvrCfg->DShortName, DPCName))
		{
			NWSAlert(12, 40, NUTHandle,
				InxMSG("Do not use DPC as the Interface", 685))
;
			goto retry;
		}
		tmpDloCfg->net_interface = interface;
	}

	NWSDestroyForm(NUTHandle);
	NWSPopList(NUTHandle);
	return K_SELECT;
}

LONG	ModifyLoginScript(FIELD *fp, int key, int *changed, NUTInfo *handle)
{
	int i;
```

```
	DloCfg_t	*tmpDloCfg;
	LONG save;

	key = key;
	changed = changed;
	handle = handle;

	tmpDloCfg = (DloCfg_t *)fp->customData;

	if (NWSPushList(NUTHandle) == 0)
		return K_SELECT;

	NWSInitForm(NUTHandle);

	i = 0;

	NWSAppendCommentField(i, 2, MSG("Wait1: ", 581), NUTHandle);
	NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_1,
				printables, F_NO_HELP, NUTHandle);

	NWSAppendCommentField(i, 40, MSG("Wait Timeout 1: ", 599), NUTHandle);
	NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfg->wait_timeo
ut_1, 2, 60, F_NO_HELP, NUTHandle);

	i++;
	NWSAppendCommentField(i, 2, MSG("Send1: ", 518), NUTHandle);
	NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_1,
				printables, F_NO_HELP, NUTHandle);

	i++;
	NWSAppendCommentField(i, 2, MSG("Wait2: ", 520), NUTHandle);
	NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_2,
				printables, F_NO_HELP, NUTHandle);
	NWSAppendCommentField(i, 40, MSG("Wait Timeout 2: ", 600), NUTHandle);
	NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfg->wait_timeo
ut_2, 2, 60, F_NO_HELP, NUTHandle);

	i++;
	NWSAppendCommentField(i, 2, MSG("Send2: ", 522), NUTHandle);
	NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_2,
				printables, F_NO_HELP, NUTHandle);

	i++;
	NWSAppendCommentField(i, 2, MSG("Wait3: ", 524), NUTHandle);
	NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_3,
				printables, F_NO_HELP, NUTHandle);
	NWSAppendCommentField(i, 40, MSG("Wait Timeout 3: ", 601), NUTHandle);
	NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfg->wait_timeo
ut_3, 2, 60, F_NO_HELP, NUTHandle);

	i++;
	NWSAppendCommentField(i, 2, MSG("Send3: ", 526), NUTHandle);
	NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_3,
				printables, F_NO_HELP, NUTHandle);

	i++;
	NWSAppendCommentField(i, 2, MSG("Wait4: ", 528), NUTHandle);
	NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_4,
				printables, F_NO_HELP, NUTHandle);
	NWSAppendCommentField(i, 40, MSG("Wait Timeout 4: ", 602), NUTHandle);
	NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfg->wait_timeo
ut_4, 2, 60, F_NO_HELP, NUTHandle);

	i++;
	NWSAppendCommentField(i, 2, MSG("Send4: ", 530), NUTHandle);
	NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_4,
                        printables, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Wait5: ", 532), NUTHandle);
        NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_5,
                        printables, F_NO_HELP, NUTHandle);
        NWSAppendCommentField(i, 40, MSG("Wait Timeout 5: ", 603), NUTHandle);
        NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfg->wait_timeo
ut_5, 2, 60, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Send5: ", 534), NUTHandle);
        NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_5,
                        printables, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Wait6: ", 536), NUTHandle);
        NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_6,
                        printables, F_NO_HELP, NUTHandle);
        NWSAppendCommentField(i, 40, MSG("Wait Timeout 6: ", 604), NUTHandle);
        NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfg->wait_timeo
ut_6, 2, 60, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Send6: ", 538), NUTHandle);
        NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_6,
                        printables, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Wait7: ", 540), NUTHandle);
        NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_7,
                        printables, F_NO_HELP, NUTHandle);
        NWSAppendCommentField(i, 40, MSG("Wait Timeout 7: ", 605), NUTHandle);
        NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfg->wait_timeo
ut_7, 2, 60, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Send7: ", 542), NUTHandle);
        NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_7,
                        printables, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Wait8: ", 544), NUTHandle);
        NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_8,
                        printables, F_NO_HELP, NUTHandle);
        NWSAppendCommentField(i, 40, MSG("Wait Timeout 8: ", 606), NUTHandle);
        NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfg->wait_timeo
ut_8, 2, 60, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Send8: ", 546), NUTHandle);
        NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_8,
                        printables, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Wait9: ", 548), NUTHandle);
        NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_9,
                        printables, F_NO_HELP, NUTHandle);
        NWSAppendCommentField(i, 40, MSG("Wait Timeout 9: ", 607), NUTHandle);
        NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfg->wait_timeo
ut_9, 2, 60, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Send9: ", 550), NUTHandle);
        NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_9,
                        printables, F_NO_HELP, NUTHandle);
```

```
        i++;

        save = NWSEditPortalForm(InxMSG("Login Script Editor", 582),
                12, 40,
/* center line, column  */
                i, 78,
/* form height, width   */
                F_NOVERIFY, F_NO_HELP,  /* Control flags, help message */
                InxMSG("Save Changes?", 583),
                NUTHandle);                                     /* Confirm message, hand
le */

//      if (save)
//      {
//              CMovB(tmpDloCfg->wait_for_1, DloCfg.wait_for_1, 30*18);
//              CMovB(&tmpDloCfg->wait_timeout_1, &DloCfg.wait_timeout_1, 9 * si
zeof(LONG));
//              DPCUpdateConfigFile();
//      }
        NWSDestroyForm(NUTHandle);
        NWSPopList(NUTHandle);
        return K_SELECT;
}

int NewProtocolFlag = -1;

LONG    ChangeProtocol(FIELD *fp, int key, int *changed, NUTInfo *handle)
{
        DloCfg_t        *tmpDloCfg;
        LIST    *listPtr, *slipList, *pppList, *netList;
        LONG ccode;
        LONG rcode = K_SELECT;

        key = key;
        changed = changed;
        handle = handle;

        tmpDloCfg = (DloCfg_t *)fp->customData;

        if (NWSPushList(NUTHandle) == 0)
                return rcode;

        NWSInitList(NUTHandle, NULL);

        slipList = NWSAppendToList(MSG("Modem - SLIP", 519), (void *)OUT_SLIP, N
UTHandle);
        pppList = NWSAppendToList(MSG("Modem - PPP", 521), (void *)OUT_PPP, NUTH
andle);
        netList = NWSAppendToList(MSG("LAN/WAN", 537), (void *)OUT_NETWORK, NUTH
andle);

        if (tmpDloCfg->out_protocol == OUT_SLIP)
        {
                listPtr = slipList;
        }
        else if (tmpDloCfg->out_protocol == OUT_PPP)
        {
                listPtr = pppList;
        }
        else
        {
                listPtr = netList;
        }

        ccode = NWSList(
                        InxMSG("Outbound Protocol", 509),
                        12, 40,
                        3,                                      /* Height */
                        16,                                     /* Width */
                        M_ESCAPE | M_SELECT,
                        &listPtr,
                        NUTHandle, NULL,
                        NULL, NULL);

        if (ccode == M_SELECT)
        {
                tmpDloCfg->out_protocol = NewProtocolFlag = (int)listPtr->otherI
nfo;
                rcode = K_ESCAPE;
        }

        NWSDestroyList(NUTHandle);
        NWSPopList(NUTHandle);
        return rcode;
}

//LONG  ChangeTunnel(FIELD *fp, int key, int *changed, NUTInfo *handle)
//{
//      DloCfg_t        *tmpDloCfg;
//      LIST    *listPtr, *enableList, *disableList;
//      LONG ccode;
//      LONG rcode = K_SELECT;
//
//      key = key;
//      changed = changed;
//      handle = handle;
//
//      tmpDloCfg = (DloCfg_t *)fp->customData;
//
//      if (NWSPushList(NUTHandle) == 0)
//              return rcode;
//
//      NWSInitList(NUTHandle, NULL);
//
//      enableList = NWSAppendToList(MSG("Enabled", 624), (void *)1, NUTHandle);
//      disableList = NWSAppendToList(MSG("Disabled", 625), (void *)0, NUTHandle
);
//
//      if (tmpDloCfg->tunnel)
//      {
//              listPtr = enableList;
//      }
//      else
//      {
//              listPtr = disableList;
//      }
//
//      ccode = NWSList(
//                      InxMSG("Tunnel Header", 626),
//                      12, 40,
//                      2,                                      /* Height */
//                      16,                                     /* Width */
//                      M_ESCAPE | M_SELECT,
//                      &listPtr,
//                      NUTHandle, NULL,
//                      NULL, NULL);
//
//      if (ccode == M_SELECT)
//      {
//              tmpDloCfg->tunnel = NewProtocolFlag = (int)listPtr->otherInfo;
//              rcode = K_ESCAPE;
```

```
//      }
//
//      NWSDestroyList(NUTHandle);
//      NWSPopList(NUTHandle);
//      return rcode;
//}

void ProviderConfiguration(void)
{
        int     i;
        void (*oldSortFunction)(LIST *, LIST *, NUTInfo *);
/*      int     save; */
        DloCfg_t        tmpDloCfg;
        int     ip0, ip1, ip2, ip3;
        int     gw0, gw1, gw2, gw3;
        MFCONTROL *mfctl0;
        int baud;
        FIELD *fp;
        char *protocolStr;
        char *pppConfigStr;
//      int cflags = F_VERIFY;
//      char *tunnelStr;

        CMovB(&DloCfg, &tmpDloCfg, sizeof(DloCfg_t));

NewProtocolLoop:
        NewProtocolFlag = -1;

        NWSInitForm(NUTHandle);

        NWSGetListSortFunction(NUTHandle, &oldSortFunction);
        NWSSetListSortFunction(NUTHandle, NoSortHandler);

        i = 0;
        NWSAppendCommentField(i, 30, MSG("Outbound Protocol : ", 525), NUTHandle
);

        if (tmpDloCfg.out_protocol == OUT_SLIP)
                protocolStr = MSG("Modem - SLIP", 527);
        else if (tmpDloCfg.out_protocol == OUT_PPP)
                protocolStr = MSG("Modem - PPP", 529);
        else
                protocolStr = MSG("LAN/WAN", 584);

        fp = NWSAppendHotSpotField(i, 50, NORMAL_FIELD, protocolStr,
                                        ChangeProtocol, NUTHandle);
        fp->customData = &tmpDloCfg;

        i+=2;
        NWSAppendCommentField(i, 2, MSG("Internet Phone           : ", 126), NU
THandle);
        NWSAppendStringField(i, 30, 30, NORMAL_FIELD, tmpDloCfg.tinet_phone_num,

                                dial_chars, F_NO_HELP, NUTHandle);

        i++;
        baud = tmpDloCfg.tinet_baud_index;
        NWSAppendCommentField(i, 2, MSG("Internet Baud            : ", 128), NU
THandle);
        mfctl0 = NWSInitMenuField(InxMSG("Baud Rate", 129), 10, 40, BaudRateHand
ler, NUTHandle);
        NWSAppendToMenuField(mfctl0, InxMSG("2400", 130), 0, NUTHandle);
        NWSAppendToMenuField(mfctl0, InxMSG("3600", 131), 1, NUTHandle);
        NWSAppendToMenuField(mfctl0, InxMSG("4800", 132), 2, NUTHandle);
        NWSAppendToMenuField(mfctl0, InxMSG("7200", 133), 3, NUTHandle);
        NWSAppendToMenuField(mfctl0, InxMSG("9600", 146), 4, NUTHandle);
        NWSAppendToMenuField(mfctl0, InxMSG("19200", 297), 5, NUTHandle);
        NWSAppendToMenuField(mfctl0, InxMSG("38400", 300), 6, NUTHandle);
        NWSAppendToMenuField(mfctl0, InxMSG("57600", 304), 7, NUTHandle);
        NWSAppendToMenuField(mfctl0, InxMSG("115200", 305), 8, NUTHandle);
        NWSAppendMenuField(i, 30, NORMAL_FIELD, &baud, mfctl0, NULL, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Internet MTU                   : ", 309), NU
THandle);
        NWSAppendIntegerField(i, 30, NORMAL_FIELD, (int *)&tmpDloCfg.mtu, 1, 655
35, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Internet Inactivity(sec)   : ", 310), NU
THandle);
        NWSAppendIntegerField(i, 30, NORMAL_FIELD, (int *)&tmpDloCfg.tinet_inact
ivity_timer, 1, 65535, F_NO_HELP, NUTHandle);

//      i++;
//      NWSAppendCommentField(i, 2, MSG("IP to IP tunneling         : ", 537), NU
THandle);
////    NWSAppendBoolField(i, 30, NORMAL_FIELD, (BYTE *)&tmpDloCfg.tunnel, F_NO_
HELP, NUTHandle);
//      if (tmpDloCfg.tunnel)
//              tunnelStr = MSG("Enabled", 627);
//      else
//              tunnelStr = MSG("Disabled", 628);
//
//      fp = NWSAppendHotSpotField(i, 30, NORMAL_FIELD, tunnelStr,
//                                      ChangeTunnel, NUTHandle);
//      fp->customData = &tmpDloCfg;
//
//      if (tmpDloCfg.tunnel)
        {
                LONG ip_address = ntohl(tmpDloCfg.ip_address);
                LONG gateway_address = ntohl(tmpDloCfg.gateway_address);
                i++;
                ip0 = (ip_address) & 0xff;
                ip1 = (ip_address >> 8) & 0xff;
                ip2 = (ip_address >> 16) & 0xff;
                ip3 = (ip_address >> 24) & 0xff;
                NWSAppendCommentField(i, 2, MSG("IP Address(ISP)            :
.   .   .   ", 306), NUTHandle);
                NWSAppendIntegerField(i, 30, NORMAL_FIELD, &ip3, 1, 255, F_NO_HE
LP, NUTHandle);
                NWSAppendIntegerField(i, 34, NORMAL_FIELD, &ip2, 1, 255, F_NO_HE
LP, NUTHandle);
                NWSAppendIntegerField(i, 38, NORMAL_FIELD, &ip1, 1, 255, F_NO_HE
LP, NUTHandle);
                NWSAppendIntegerField(i, 42, NORMAL_FIELD, &ip0, 1, 255, F_NO_HE
LP, NUTHandle);

                i++;
                gw0 = (gateway_address) & 0xff;
                gw1 = (gateway_address >> 8) & 0xff;
                gw2 = (gateway_address >> 16) & 0xff;
                gw3 = (gateway_address >> 24) & 0xff;
                NWSAppendCommentField(i, 2, MSG("Hybrid Gateway Address     :
.   .   .   ", 307), NUTHandle);
                NWSAppendIntegerField(i, 30, NORMAL_FIELD, &gw3, 1, 255, F_NO_HE
LP, NUTHandle);
                NWSAppendIntegerField(i, 34, NORMAL_FIELD, &gw2, 1, 255, F_NO_HE
LP, NUTHandle);
                NWSAppendIntegerField(i, 38, NORMAL_FIELD, &gw1, 1, 255, F_NO_HE
LP, NUTHandle);
                NWSAppendIntegerField(i, 42, NORMAL_FIELD, &gw0, 1, 255, F_NO_HE
```

```
LP, NUTHandle);
        }


        i+=2;
//      NWSAppendCommentField(i, 30, MSG("Slip Configuration", 552), NUTHandle);

        NWSAppendCommentField(i, 2, MSG("Auto Login Enabled         : ", 553), NU
THandle);
        NWSAppendBoolField(i, 30, NORMAL_FIELD, (BYTE *)&tmpDloCfg.auto_login, F
_NO_HELP, NUTHandle);

        i++;
        protocolStr = MSG("Modify Auto Login Script", 535);

        fp = NWSAppendHotSpotField(i, 25, NORMAL_FIELD, protocolStr,
                                        ModifyLoginScript, NUTHandle);
//      fp->customDataRelease = NWSFree;
        fp->customData = &tmpDloCfg;

        i++;
        if (tmpDloCfg.out_protocol == OUT_PPP)
        {
                pppConfigStr = MSG("Modify PPP Configuration", 608);

                fp = NWSAppendHotSpotField(i, 26, NORMAL_FIELD, pppConfigStr,
                                                ModifyPPPConfig, NUTHandle);
                fp->customData = &tmpDloCfg;
        }
        else if (tmpDloCfg.out_protocol == OUT_NETWORK)
        {
                fp = NWSAppendHotSpotField(i, 26, NORMAL_FIELD,
                                        "Modify Network Configuration",
                                        ModifyNetConfig, NUTHandle);
                fp->customData = &tmpDloCfg;
        }

        i++;

        /* save =*/ NWSEditPortalForm(InxMSG("Provider Configuration Editor", 55
5),
                12, 40,
/* center line, column  */
                i, 78,
/* form height, width   */
                /* cflags*/F_NOVERIFY, F_NO_HELP,
tr
ol flags, help message */
                /* InxMSG("Save Changes?", 556)*/ NULL,
                NUTHandle);                                     /* Confirm message, hand
le */

        NWSSetListSortFunction(NUTHandle, oldSortFunction);
        NWSDestroyForm(NUTHandle);

        if (NewProtocolFlag != -1)
        {
//              cflags = F_FORCE;
                goto NewProtocolLoop;
        }

//      if (!save)
//              return;

        tmpDloCfg.ip_address = htonl((ip0) |
                                (ip1 << 8) |
                                (ip2 << 16) |
                                        (ip3 << 24));

        tmpDloCfg.gateway_address = htonl((gw0) |
                                          (gw1 << 8) |
                                          (gw2 << 16) |
                                          (gw3 << 24));

        tmpDloCfg.tinet_baud_index = baud;

        if (memcmp(&DloCfg, &tmpDloCfg, sizeof(DloCfg)) == 0 ||
            NWSConfirm(InxMSG("Save Changes?", 612), 0, 0, TRUE, NULL,
                       NUTHandle, NULL) == FALSE)
          return;

        /*
         * Get newest wait_for and send strings in case ModifyLoginScript()
         * changed them.
         */
//      CMovB(DloCfg.wait_for_1, tmpDloCfg.wait_for_1, 30 * 18);
//      CMovB(&DloCfg.wait_timeout_1, &tmpDloCfg.wait_timeout_1, 9 * sizeof(LONG
));

        CMovB(&tmpDloCfg, &DloCfg, sizeof(DloCfg_t));

        InetChangeProtocol();

        DPCUpdateConfigFile();
}

void ModemConfiguration(void)
{
        int     i;
        int     save;
        DloCfg_t        tmpDloCfg;
        void (*oldSortFunction)(LIST *, LIST *, NUTInfo *);

        CMovB(&DloCfg, &tmpDloCfg, sizeof(DloCfg_t));

        NWSInitForm(NUTHandle);

        NWSGetListSortFunction(NUTHandle, &oldSortFunction);
        NWSSetListSortFunction(NUTHandle, NoSortHandler);

        i = 0;
        NWSAppendCommentField(i, 2, MSG("Packet Life(sec)    : ", 325), NUTHandl
e);
        NWSAppendIntegerField(i, 24, NORMAL_FIELD, (int *)&tmpDloCfg.packet_life
time, 1, 65535, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Call Setup(sec)     : ", 326), NUTHandl
e);
        NWSAppendIntegerField(i, 24, NORMAL_FIELD, (int *)&tmpDloCfg.call_setup_
timeout, 1, 65535, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Aync Buffer Size    : ", 212), NUTHandl
e);
        NWSAppendIntegerField(i, 24, NORMAL_FIELD, (int *)&tmpDloCfg.async_buffe
r_size, 1500, 1024*100, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Dial Prefix         : ", 329), NUTHandl
e);
        NWSAppendStringField(i, 24, 10, NORMAL_FIELD, tmpDloCfg.dialout_prefix,
```

```
                                modem_control_chars, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Hangup Str          : ", 330), NUTHandl
e);
        NWSAppendStringField(i, 24, 20, NORMAL_FIELD, tmpDloCfg.hangup_str,
                                modem_control_chars, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Disconnect str      : ", 331), NUTHandl
e);
        NWSAppendStringField(i, 24, 20, NORMAL_FIELD, tmpDloCfg.disconnect_str,
                                modem_control_chars, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Escape Str          : ", 332), NUTHandl
e);
        NWSAppendStringField(i, 24, 20, NORMAL_FIELD, tmpDloCfg.escape_str,
                                modem_control_chars, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Connect str         : ", 333), NUTHandl
e);
        NWSAppendStringField(i, 24, 50, NORMAL_FIELD, tmpDloCfg.connect_str,
                                modem_control_chars, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Init str            : ", 334), NUTHandl
e);
        NWSAppendStringField(i, 24, 60, NORMAL_FIELD, tmpDloCfg.init_str,
                                modem_control_chars, F_NO_HELP, NUTHandle);

        i++;

        save = NWSEditPortalForm(InxMSG("Modem Configuration Editor", 513),
                12, 40,
/* center line, column  */
                i, 76,
/* form height, width   */
                F_VERIFY, F_NO_HELP,                    /* Control flags, help m
essage */
                InxMSG("Save Changes?", 514),
                NUTHandle);                             /* Confirm message, hand
le */

        NWSSetListSortFunction(NUTHandle, oldSortFunction);
        NWSDestroyForm(NUTHandle);

        if (!save)
                return;

        CMovB(&tmpDloCfg, &DloCfg, sizeof(DloCfg_t));

        if (AIOPortHandle != -1)
        {
                AIOSetWriteBufferSize(AIOPortHandle, DloCfg.async_buffer_size);
                AIOGetWriteBufferSize(AIOPortHandle, &AIOWriteBufferSize);
                DloPMaxBufferSize = (AIOWriteBufferSize < DLOBUFSIZE) ? AIOWrite
BufferSize : DLOBUFSIZE;
                DloIMaxBufferSize = (AIOWriteBufferSize < DLOBUFSIZE) ? AIOWrite
BufferSize : DLOBUFSIZE;
        }

        DPCUpdateConfigFile();
}
```

```
void DPCConfiguration(void)
{
        LIST    *listPtr = NULL;
        LONG ccode = M_SELECT;

        NWSInitList(NUTHandle, NULL);

        NWSAppendToList(MSG("Modem Configuration", 263), (void *)1, NUTHandle);
        NWSAppendToList(MSG("Package Delivery Configuration", 507), (void *)2, N
UTHandle);
        NWSAppendToList(MSG("Provider Configuration", 508), (void *)3, NUTHandle
);
        while (ccode != M_ESCAPE)
        {
                ccode = NWSList(
                                InxMSG("DPC Configuration", 533),
                                12, 40,
                                3,                                      /* Heigh
t */
                                32,                                     /* Width
 */
                                M_ESCAPE | M_SELECT,
                                &listPtr,
                                NUTHandle, NULL,
                                NULL, NULL);

                if (ccode == M_SELECT)
                {
                        if (NWSPushList(NUTHandle) != 0)
                        {
                                switch ((int)listPtr->otherInfo)
                                {
                                        case 1:
                                                ModemConfiguration();
                                                break;

                                        case 2:
                                                PDConfiguration();
                                                break;

                                        case 3:
                                                ProviderConfiguration();
                                                break;

                                        default:
                                                break;
                                }
                                NWSPopList(NUTHandle);
                        }
                }
        }
        NWSDestroyList(NUTHandle);


}
/**************************************************************************
*
*       DloReceive(BYTE *buffer
*                   int size)
*
*       Description:
*               This routine reads a bytes from the global modem receive buffer
*               and returns it to the caller.
*
*       Input:
```

```
*               size                                    - max number of bytes to
 read
*
*       Output:
*               buffer                                  - buffer to write data into
*
*       Returns:
*               Number of bytes read
*
***************************************************************************/

int DloGetRcvBytesBuffered(void)
{
        return(DloRcvCount);
}

int DloReceive (BYTE *buffer, int size)
{
        int i,iRetSize = 0;
        if(DloRcvCount)
        {
                iRetSize = (DloRcvCount > size)? size : DloRcvCount;

                DloRcvCount -= iRetSize;

                for(i=0;i<iRetSize;i++)
                {
                        buffer[i] = DloRcvBuffer[DloReadIndex++];
                        if(DloReadIndex >= DLOBUFSIZE)
                                DloReadIndex = 0;
                }
        }

        return iRetSize;
}

/**************************************************************************
*
*       DloEmpty(void)
*
*       Description:
*               This routine returns true if the modem transmit buffer is empty,
*               meaning that all previous requests have been sent.
*
*       Input:
*               nothing
*
*       Output:
*               nothing
*
*       Returns:
*               TRUE if transmit buffer is empty
*
***************************************************************************/

int DloEmpty (void)
{
        if (DloConn == DLO_CONN_PACKAGE)
                return (DloPXmitCount == 0);
        else
                return (DloIXmitCount == 0);
}

/**************************************************************************
*
*       WriteCommPhoneNumber(void)
```

```
*
*       Description:
*               This routine is called when the modem needs to be dailed.
*
*       Input:
*               nothing
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
*************************************************************************/

static void WriteCommPhoneNumber(void)
{
        int baudIndex, i;
        char *number;

        if (DloConn == DLO_CONN_PACKAGE)
                baudIndex = DloCfg.pdeliv_baud_index;
        else
                baudIndex = DloCfg.tinet_baud_index;

        AIOConfigurePort(AIOPortHandle,
                        AIOBaudRateDefines[baudIndex],
                        AIO_DATA_BITS_8, AIO_STOP_BITS_1,
                        AIO_PARITY_NONE,
                        AIO_SOFTWARE_FLOW_CONTROL_OFF | AIO_HARDWARE_FLOW_CONTRO
L_ON);

        if (DloCfg.dialout_prefix[0])
        {
                if (DloConn == DLO_CONN_PACKAGE)
                        number = DloCfg.pdeliv_phone_num;
                else
                        number = DloCfg.tinet_phone_num;
                for (i = 0; number[i]; i++)
                {
                        if (number[i] < 'A' || number[i] > 'z')
                                break;
                        if (number[i] > 'Z' && number[i] < 'a')
                                break;
                }
                if (i)
                        SendAIOData(number, i);         /* Send the alpha string
 */

                SendAIOData(DloCfg.dialout_prefix, CStrLen(DloCfg.dialout_prefix
));
                SendAIOData(&number[i], CStrLen(&number[i]));
        }
        else
        {
                if (DloConn == DLO_CONN_PACKAGE)
                        SendAIOData(DloCfg.pdeliv_phone_num, CStrLen(DloCfg.pdel
iv_phone_num));
                else
                        SendAIOData(DloCfg.tinet_phone_num, CStrLen(DloCfg.tinet
_phone_num));
        }

        SendAIOData(MSG("\r", 613), 1);

        if (DloState == DLOS_REDL)
```

```
                UpdateModemStr(MSG("Modem Status: Redialing
               \n", 559));
        else
                UpdateModemStr(MSG("Modem Status: Dialing
               \n", 560));
}

/*************************************************************************
*
*       ProcessDisconnect(void)
*
*       Description:
*               This routine is called when where attaching and we lose Carrier
*               Detect.
*
*       Input:
*               nothing
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
*************************************************************************/


static void ProcessDisconnect(void)
{
        int     count;

        if (DloConn == DLO_CONN_PACKAGE)
        {
                UpdateModemStr(MSG("Modem Status: Disconnected from Package Deli
very           \n", 237));
                count = DloPXmitCount;
        }
        else
        {
                UpdateModemStr(MSG("Modem Status: Disconnected from Internet
               \n", 472));
                count = DloIXmitCount;
        }

        if (count)
        {
                WriteCommPhoneNumber();
                DloStartTimer(DloCfg.call_setup_timeout * 19);
                DloState = DLOS_DIAL;
                if (DloConn == DLO_CONN_INET)
                        InetStateChange(DLOS_DIAL);
        }
        else
        {
                DloStartTimer(1 * 19);
                DloState = DLOS_DISC_4;
                if (DloConn == DLO_CONN_INET)
                        InetStateChange(DLOS_DISC_4);
        }
}

/*************************************************************************
*
*       DloEndConn(void)
*
*       Description:
```

```
*               This routine is terminate a modem connection.
*
*       Input:
*               nothing
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
****************************************************************************/

void DloEndConn(void)
{
        if (AIOPortHandle < 0)
                return;

        if (DloConn == DLO_CONN_PACKAGE)
                DloPXmitCount = 0;
        else
                DloIXmitCount = 0;

        switch(DloState)
        {
                case DLOS_INIT:
                case DLOS_REDL:
                case DLOS_DIAL:
                case DLOS_CONN:
                        AIOFlushBuffers(AIOPortHandle, (AIO_FLUSH_WRITE_BUFFER |
 AIO_FLUSH_READ_BUFFER));
                        DloState = DLOS_CONN;
                        StateMachine(DLOE_TIMEOUT);
                        break;
                case DLOS_IDLE:
                case DLOS_DISC_1:
                case DLOS_DISC_2:
                case DLOS_DISC_3:
                case DLOS_DISC_4:
                  StateMachine(DLOE_DISCONN);
                  break;
        }

}
/***************************************************************************
*
*       _0003(void)                     In IDLE state, got SND event
*
*       Description:
*               The state machine is in the IDLE state.
*               We've received a SND request.
*
*       Input:
*               nothing
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
****************************************************************************/

int     DloConnected(void)
{
```

```
        LONG    extStatus = 0, chgdExtStatus;

        if (AIOPortHandle < 0)
                return(FALSE);

        AIOGetExternalStatus( AIOPortHandle, &extStatus, &chgdExtStatus);
        if (extStatus & AIO_EXTSTA_DCD)
                return(TRUE);

        return(FALSE);

}

static void _0003 (void)
{
        LONG    extStatus, chgdExtStatus;

        InitializeAIO();
        if (AIOPortHandle < 0)
        {
                UpdateModemStr(MSG("Modem Status: ERROR - Unable to initialize A
IO             \n", 238));
                return;
        }

        if (AIOGetExternalStatus( AIOPortHandle, &extStatus, &chgdExtStatus)
                        || !(extStatus & AIO_EXTSTA_DCD))
        {
                AIOFlushBuffers(AIOPortHandle,
                                (AIO_FLUSH_WRITE_BUFFER | AIO_FLUSH_READ_BUFFER)
);
                SendAIOData(DloCfg.init_str, CStrLen(DloCfg.init_str));
                SendAIOData(MSG("\r", 614), 1);
                DloStartTimer(5  * 19);
                DloState = DLOS_INIT;
                if (DloConn == DLO_CONN_INET)
                        InetStateChange(DLOS_INIT);
                UpdateModemStr(MSG("Modem Status: Initializing Modem
               \n", 561));
        }
        else
        {
                if (DloConn == DLO_CONN_PACKAGE)
                        DloStartTimer(DloPInactivityTimer);
                else
                        DloStartTimer(DloIInactivityTimer);
//              DloStartTimer(30*19);
                DloStopPacketLifeTimer();

                WriteCommXmitBuffer();
                DloState = DLOS_CONN;
                if (DloConn == DLO_CONN_INET)
                        InetStateChange(DLOS_CONN);
                if (DloConn == DLO_CONN_PACKAGE)
                        UpdateModemStr(MSG("Modem Status: Connected to Package D
elivery                \n", 562));
                else
//                      UpdateModemStr(MSG("Modem Status: Connected to Internet
                       \n", 473));
                {
                        if (ConnectBaudStr[0])
                        {
                                BYTE connectStr[80];

                                NWsprintf(connectStr, MSG("Modem Status: Connect
ed to Internet at%.24s\n", 563), ConnectBaudStr);
```

```
                                        UpdateModemStr(connectStr);
                                }
                                else
                                {
                                        UpdateModemStr(MSG("Modem Status: Connected to I
nternet                                \n", 564));
                                }
                        }
                }
}

/***************************************************************************
 *
 *      _0100(void)                     In INIT state, got TIMEOUT
 *
 *      Description:
 *              The state machine is in the INIT state.
 *              We timed out waiting for the modem init sequence.
 *
 *      Input:
 *              nothing
 *
 *      Output:
 *              nothing
 *
 *      Returns:
 *              nothing
 *
 ***************************************************************************/

static void _0100(void)
{
        LONG    extStatus, chgdExtStatus;

        if (AIOGetExternalStatus( AIOPortHandle, &extStatus, &chgdExtStatus)
                        || !(extStatus & AIO_EXTSTA_DCD))
        {
                WriteCommPhoneNumber();
                DloStartTimer(DloCfg.call_setup_timeout * 19);
                DloState = DLOS_DIAL;
                if (DloConn == DLO_CONN_INET)
                        InetStateChange(DLOS_DIAL);
        }
        else
        {
                UpdateModemStr(MSG("Modem Status: Error - Still Connected
               \n", 242));
                DloEndConn();
        }
}

/***************************************************************************
 *
 *      _0104(void)                     In INIT state, got OK response from mode
m
 *
 *      Description:
 *              The state machine is in the INIT state.
 *              We've received an OK response from sending modem init sequence.
 *
 *      Input:
 *              nothing
 *
 *      Output:
 *              nothing
 *
```

```
 *      Returns:
 *              nothing
 *
 ***************************************************************************/

static void _0104(void)
{
        WriteCommPhoneNumber();

        DloStartTimer(DloCfg.call_setup_timeout * 19);
        DloState = DLOS_DIAL;
        if (DloConn == DLO_CONN_INET)
                InetStateChange(DLOS_DIAL);
}

/***************************************************************************
 *
 *      _0200(void)                     In DIAL state, got TIMEOUT
 *
 *      Description:
 *              The state machine is in the DIAL state.
 *              It timed out waiting for connect.
 *
 *      Input:
 *              nothing
 *
 *      Output:
 *              nothing
 *
 *      Returns:
 *              nothing
 *
 ***************************************************************************/

static void _0200(void)
{
        AIOFlushBuffers(AIOPortHandle, AIO_FLUSH_WRITE_BUFFER);
        AIOFlushBuffers(AIOPortHandle, AIO_FLUSH_READ_BUFFER);
        SendAIOData(MSG("\r", 615), 1);
        DloStartTimer(10 * 18);
        DloState = DLOS_REDL;
        if (DloConn == DLO_CONN_INET)
                InetStateChange(DLOS_REDL);
}

/***************************************************************************
 *
 *      _0201(void)                     In DIAL state, got CONNECT response from
 modem
 *
 *      Description:
 *              The state machine is in the DIAL state.
 *              We've received a CONNECT response from sending ATDT sequence.
 *
 *      Input:
 *              nothing
 *
 *      Output:
 *              nothing
 *
 *      Returns:
 *              nothing
 *
 ***************************************************************************/

static void _0201(void)
```

```
{
        WriteCommXmitBuffer();
        if (DloConn == DLO_CONN_PACKAGE)
        {
                DloStartTimer(DloPInactivityTimer);
                UpdateModemStr(MSG("Modem Status: Connected to Package Delivery
               \n", 244));
        }
        else
        {
                DloStartTimer(DloIInactivityTimer);
//              UpdateModemStr(MSG("Modem Status: Connected to Internet
               \n", 474));
                if (ConnectBaudStr[0])
                {
                        BYTE connectStr[80];

                        NWsprintf(connectStr, MSG("Modem Status: Connected to In
ternet at%.24s\n", 491), ConnectBaudStr);
                        UpdateModemStr(connectStr);
                }
                else
                {
                        UpdateModemStr(MSG("Modem Status: Connected to Internet
                       \n", 492));
                }
        }

        DloStopPacketLifeTimer();
        DloState = DLOS_CONN;
        if (DloConn == DLO_CONN_INET)
                InetStateChange(DLOS_CONN);
}

/****************************************************************************
*
*       _0202(void)                     In DIAL state, got disconnected
*
*       Description:
*               The state machine is in the DIAL state.
*               We got disconnected from remote side.
*
*       Input:
*               nothing
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
****************************************************************************/

static void _0202(void)
{
        DloEndConn();
}

/****************************************************************************
*
*       _0205(void)                     In DIAL state, got BUSY response from mo
dem
*
*       Description:
*               The state machine is in the DIAL state.
*               We've received a BUSY response from sending ATDT sequence.
```

```
*
*       Input:
*               nothing
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
****************************************************************************/

static void _0205(void)
{
        UpdateModemStr(MSG("Modem Status: Busy
       \n", 245));
}

/****************************************************************************
*
*       _0206(void)                     In DIAL state, got NODIALTONE response f
rom modem
*
*       Description:
*               The state machine is in the DIAL state.
*               We've received an NODIALTONE response from sending ATDT sequence
.
*
*       Input:
*               nothing
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
****************************************************************************/

static void _0206(void)
{
        UpdateModemStr(MSG("Modem Status: No Dialtone
       \n", 246));
        DloStartTimer(5 * 19);
}

/****************************************************************************
*
*       _0207(void)                     In DIAL state, got RING response from mo
dem
*
*       Description:
*               The state machine is in the DIAL state.
*               We've received a RING response from sending modem ATDT sequence.
*
*       Input:
*               nothing
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
****************************************************************************/
```

```
static void _0207(void)
{
        UpdateModemStr(MSG("Modem Status: Ringing!!!
       \n", 247));
}

/************************************************************************
*
*       _0208(void)                       In DIAL state, got NO ANSWER response fr
om modem
*
*       Description:
*               The state machine is in the DIAL state.
*               We've received a NO ANSWER response from sending ATDT sequence.
*
*       Input:
*               nothing
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
************************************************************************/

static void _0208(void)
{
        UpdateModemStr(MSG("Modem Status: No ANSWER
       \n", 248));
}

/************************************************************************
*
*       _0300(void)                       In REDIAL state, got TIMEOUT
*
*       Description:
*               The state machine is in the REDIAL state.
*               We timed out.
*
*       Input:
*               nothing
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
************************************************************************/

static void _0300(void)
{
        UpdateModemStr(MSG("Modem Status: Timeout - Reinitializing the modem
       \n", 249));

        AIOFlushBuffers(AIOPortHandle, AIO_FLUSH_WRITE_BUFFER);
        AIOFlushBuffers(AIOPortHandle, AIO_FLUSH_READ_BUFFER);
        SendAIOData(DloCfg.init_str, CStrLen(DloCfg.init_str));
        SendAIOData(MSG("\r", 616), 1);
        DloStartTimer(5 * 19);
        DloState = DLOS_INIT;
        if (DloConn == DLO_CONN_INET)
                InetStateChange(DLOS_INIT);
}
```

```
/************************************************************************
*
*       _0302(void)                       In REDIAL state, got Disconnected
*
*       Description:
*               The state machine is in the REDIAL state.
*               We got disconnected by the remote site.
*
*       Input:
*               nothing
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
************************************************************************/

static void _0302(void)
{
        DloEndConn();
}

/************************************************************************
*
*       _0104(void)                       In REDIAL state, got OK response from mo
dem
*
*       Description:
*               The state machine is in the REDIAL state.
*               We've received an OK response from sending modem init sequence.
*
*       Input:
*               nothing
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
************************************************************************/

static void _0304(void)
{
        WriteCommPhoneNumber();
        DloStartTimer(DloCfg.call_setup_timeout * 19);
        DloState = DLOS_DIAL;
        if (DloConn == DLO_CONN_INET)
                InetStateChange(DLOS_DIAL);
}

/************************************************************************
*
*       _0400(void)                       In CONNECTED state, timed out
*
*       Description:
*               State:  The state machine is in the CONNECTED state.
*               Event:  We timed out due to inactivity.
*               Action: Add a 1.5 sec pre-escape delay. DLOS_DISC_1 state.
*
*       Input:
*               nothing
*
*       Output:
 *              nothing
 *
 *      Returns:
 *              nothing
 *
 *****************************************************************************/

static void _0400(void)
{
        DloStartTimer( (1 * 19) + 9);           /* 1.5 second delay */
        DloState = DLOS_DISC_1;
        if (DloNextConn == DloConn)
          DloNextConn = DLO_CONN_IDLE;
        if (DloConn == DLO_CONN_INET) {
          UpdateModemStr(MSG("Modem Status: Disconnecting from Internet
         \n", 566));
          InetStateChange(DLOS_DISC_1);
        }
        else if (DloConn == DLO_CONN_PACKAGE)
          UpdateModemStr(MSG("Modem Status: Disconnecting from Package Delivery
         \n", 565));
        else
          UpdateModemStr("Modem Status: Disconnecting
     \n");
}

/*****************************************************************************
 *
 *      _0402(void)                     In CONNECT state, got disconnected
 *
 *      Description:
 *              State:  The state machine is in the CONNECT state.
 *              Event:  We were disconnected by the remote site.
 *              Action: If still have data to send:
 *                                                      Send connect string(ATDT
1800...), DLOS_DIAL state.
 *                                              else
 *                                                      Start 1 second timer, DL
OS_DISC_4 state.
 *
 *      Input:
 *              nothing
 *
 *      Output:
 *              nothing
 *
 *      Returns:
 *              nothing
 *
 *****************************************************************************/

static void _0402(void)
{
        ProcessDisconnect();
}

/*****************************************************************************
 *
 *      _0403(void)                     In CONNECTED state, got SEND request
 *
 *      Description:
 *              State:  The state machine is in the CONNECTED state.
 *              Event:  We've received a request to send data to the remote site
 .
 *              Action: Extend the inactivity timer.
 *
```

```
 *      Input:
 *              nothing
 *
 *      Output:
 *              nothing
 *
 *      Returns:
 *              nothing
 *
 *****************************************************************************/

static void _0403(void)
{
        if (DloConn == DLO_CONN_PACKAGE)
                DloStartTimer(DloPInactivityTimer);
        else
                DloStartTimer(DloIInactivityTimer);
}

/*****************************************************************************
 *
 *      _0500(void)                     In Disconnect 1 state, got timed out
 *
 *      Description:
 *              State:  Disconnect 1 state
 *                                                      Two ways to get in:
 *                                                      1)
 *                                                                    - Inactivity timer kicke
d in
 *                                                                    - Set 1.5 pre-escape tim
er(DLOS_DISC_1)
 *                                                      2)
 *                                                                - Inactivity timer kicked in
 *                                                                - Set 1.5 pre-escape timer(DLO
S_DISC_1)
 *                                                                    - Sent escape string w/2
 second timer(DLOS_DISC_2)
 *                                                                    - Sent hangup string w/1
0 second timer(DLOS_DISC_3)
 *                                                                    - Timed out
 *
 *              Event:  Intentional 1.5 or 10 second timeout.
 *              Action: Send escape string(+++) set 2 second timer, DLOS_DISC_2
state.
 *
 *      Input:
 *              nothing
 *
 *      Output:
 *              nothing
 *
 *      Returns:
 *              nothing
 *
 *****************************************************************************/

static void _0500(void)
{
        AIOFlushBuffers(AIOPortHandle,
                        (AIO_FLUSH_WRITE_BUFFER | AIO_FLUSH_READ_BUFFER));

        if (DebugFlag)
          fputs(DloCfg.escape_str, stdout);

        SendAIOData(DloCfg.escape_str, CStrLen(DloCfg.escape_str));
        DloStartTimer(2 * 19);
        DloState = DLOS_DISC_2;
        if (DloConn == DLO_CONN_INET)
                InetStateChange(DLOS_DISC_2);
}

/***************************************************************************
 *
 *      _0502(void)                     In Disconnect 1 state, got disconnected
 *
 *      Description:
 *              State: Disconnect 1 state.
 *                                      Two ways to get in:
 *                                      1)
 *                                                      - Inactivity timer kicke
d in
 *                                                      - Set 1.5 pre-escape tim
er(DLOS_DISC_1)
 *                                      2)
 *                                              - Inactivity timer kicked in
 *                                              - Set 1.5 pre-escape timer(DLO
S_DISC_1)
 *                                                      - Sent escape string w/2
 second timer(DLOS_DISC_2)
 *                                                      - Sent hangup string w/1
0 second timer(DLOS_DISC_3)
 *                                                      - Timed out
 *              Event:  We were disconnected by the remote site.
 *              Action: If still have data to send:
 *                                                      Send connect string(ATDT
1800...), DLOS_DIAL state.
 *                                              else
 *                                                      Start 1 second timer, DL
OS_DISC_4 state.
 *
 *      Input:
 *              nothing
 *
 *      Output:
 *              nothing
 *
 *      Returns:
 *              nothing
 *
 ***************************************************************************/

static void _0502(void)
{
        ProcessDisconnect();
}

/***************************************************************************
 *
 *      _0600(void)                     In Disconnect 2 state, timed out
 *
 *      Description:
 *              State:  Disconnect 2 state:
 *                                              - Inactivity timer kicked in
 *                                              - Set 1.5 pre-escape timer(DLO
S_DISC_1)
 *                                                      - Sent escape string w/2
 second timer(DLOS_DISC_2)
 *              Event:  We timed out, modem didn't respond.
 *              Action: Send Hangup string with 10 second timer, DLOS_DISC_3 sta
te.
 *
 *
```

```
 *      Input:
 *              nothing
 *
 *      Output:
 *              nothing
 *
 *      Returns:
 *              nothing
 *
 ***************************************************************************/

static void _0600(void)
{
        SendAIOData(DloCfg.hangup_str, CStrLen(DloCfg.hangup_str));
        SendAIOData(MSG("\r", 617), 1);
        delay(2000);
        DloStartTimer(10 * 19);
        DloState = DLOS_DISC_3;
        if (DloConn == DLO_CONN_INET)
                InetStateChange(DLOS_DISC_3);
}

/***************************************************************************
 *
 *      _0602(void)                     In Disconnect 2 state, got disconnected
 *
 *      Description:
 *              State:  Disconnect 2 state:
 *                                              - Inactivity timer kicked in
 *                                              - Set 1.5 pre-escape timer(DLO
S_DISC_1)
 *                                                      - Sent escape string w/2
 second timer(DLOS_DISC_2)
 *              Event:  We got disconnected by the remote site while waiting aft
er
 *                                              sending +++.
 *              Action: If still have data to send:
 *                                                      Send connect string(ATDT
1800...), DLOS_DIAL state.
 *                                              else
 *                                                      Start 1 second timer, DL
OS_DISC_4 state.
 *
 *      Input:
 *              nothing
 *
 *      Output:
 *              nothing
 *
 *      Returns:
 *              nothing
 *
 ***************************************************************************/

static void _0602(void)
{
        ProcessDisconnect();
}

/***************************************************************************
 *
 *      _0604(void)                     In Disconnect 2 state, got OK response f
rom modem
 *
 *      Description:
 *              State:  Disconnect 2 state:
```

```
 *                                               - Inactivity timer kicked in
 *                                               - Set 1.5 pre-escape timer(DLO
S_DISC_1)
 *                                                  - Sent escape string w/2
 second timer(DLOS_DISC_2)
 *              Event:  We received an OK response.
 *              Action: Send Hangup string with 10 second timer, DLOS_DISC_3 sta
te.
 *
 *      Input:
 *              nothing
 *
 *      Output:
 *              nothing
 *
 *      Returns:
 *              nothing
 *
 *************************************************************************/

static void _0604(void)
{
        SendAIOData(DloCfg.hangup_str, CStrLen(DloCfg.hangup_str));
        SendAIOData(MSG("\r", 618), 1);
        delay(5000);
        DloStartTimer(10 * 19);
        DloState = DLOS_DISC_3;
        if (DloConn == DLO_CONN_INET)
                InetStateChange(DLOS_DISC_3);
}

/************************************************************************
 *
 *      _0700(void)                     In Disconnect 3 state, timed out
 *
 *      Description:
 *              State:  Disconnect 3 state:
 *                                               - Inactivity timer kicked in
 *                                               - Set 1.5 pre-escape timer(DLO
S_DISC_1)
 *                                                  - Sent escape string w/2
 second timer(DLOS_DISC_2)
 *                                                  - Sent hangup string w/1
0 second timer(DLOS_DISC_3)
 *              Event:  We timed out.
 *              Action: Drop DTR and RTS for 1 second, and start hangup
 *                                         sequence over again(DLOS_DISC_2)
 *
 *      Input:
 *              nothing
 *
 *      Output:
 *              nothing
 *
 *      Returns:
 *              nothing
 *
 *************************************************************************/

static void _0700(void)
{
        AIOSetExternalControl(AIOPortHandle, AIO_EXTERNAL_CONTROL, 0);
        delay(1000);
        AIOSetExternalControl(AIOPortHandle,
                              AIO_EXTERNAL_CONTROL,
                              (AIO_EXTCTRL_DTR | AIO_EXTCTRL_RTS));
```

```
        delay(500);
        DloStartTimer(1 * 19);
        DloState = DLOS_DISC_2;
        if (DloConn == DLO_CONN_INET)
                InetStateChange(DLOS_DISC_2);
}

/************************************************************************
 *
 *      _0702(void)                     In Disconnect 3 state, got disconnected
 *
 *      Description:
 *              State:  Disconnect 3 state:
 *                                               - Inactivity timer kicked in
 *                                               - Set 1.5 pre-escape timer(DLO
S_DISC_1)
 *                                                  - Sent escape string w/2
 second timer(DLOS_DISC_2)
 *                                                  - Sent hangup string w/1
0 second timer(DLOS_DISC_3)
 *              Event:  We got disconnected by remote site while waiting [illegible] han
gup
 *                                          sequence.
 *              Action: If still have data to send:
 *                                                      Send connect string(ATDT
1800...), DLOS_DIAL state.
 *                                          else
 *                                                      Start 1 second timer, DL
OS_DISC_4 state.
 *
 *      Input:
 *              nothing
 *
 *      Output:
 *              nothing
 *
 *      Returns:
 *              nothing
 *
 *************************************************************************/

static void _0702(void)
{
        ProcessDisconnect();
}

/************************************************************************
 *
 *      _0704(void)                     In Disconnect 3 state, got OK response f
rom modem
 *
 *      Description:
 *              State:  Disconnect 3 state:
 *                                               - Inactivity timer kicked in
 *                                               - Set 1.5 pre-escape timer(DLO
S_DISC_1)
 *                                                  - Sent escape string w/2
 second timer(DLOS_DISC_2)
 *                                                  - Sent hangup string w/1
0 second timer(DLOS_DISC_3)
 *              Event:  We've received an OK response from sending hangup sequen
ce.
 *              Action: If still have data to send:
 *                                                      Send connect string(ATDT
1800...), DLOS_DIAL state.
 *                                          else
*                                                          Start 1 second timer, DL
OS_DISC_4 state.
*
*       Input:
*               nothing
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
*****************************************************************************/

static void _0704(void)
{
        ProcessDisconnect();
}

/*****************************************************************************
*
*       _0800(void)                       In Disconnect 4 state, got timeout
*
*       Description:
*               State:  Disconnect 4 state:
*                                                      - Inactivity timer kicked in
*                                                      - Set 1.5 pre-escape timer(DLO
S_DISC_1)
*                                                             - Sent escape string w/2
 second timer(DLOS_DISC_2)
*                                                             - Sent hangup string w/1
0 second timer(DLOS_DISC_3)
*                                                             - If nothing more to sen
d, set 1 second timer(DLOS_DISC_4)
*               Event:  We've intentionally timed out.
*               Action: Hangup is complete and there is nothing more to send.
*
*       Input:
*               nothing
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
*****************************************************************************/

static void _0800(void)
{
        if (DloNextConn == DLO_CONN_IDLE)
        {
                DloConn = DLO_CONN_IDLE;
                UpdateModemStr(MSG("Modem Status:  IDLE
               \n", 567));
                DloState = DLOS_IDLE;
                InetStateChange(DLOS_IDLE);
        }
        else
        {
                DloConn = DloNextConn;
                DloNextConn = DLO_CONN_IDLE;
                if (DloConn == DLO_CONN_INET)
                {
                        DloIInactivityTimer = DloCfg.tinet_inactivity_timer * 19
;
```

```
                }
                DloState = DLOS_IDLE;
                StateMachine(DLOE_SEND);
        }
}

static _statefn statetable[DLOENUM][DLOSNUM] =
{
//   (0)    (1)    (2)    (3)    (4)    (5)    (6)    (7)    (8)
//  IDLE   INIT   DIAL   REDL   CONN   DIS1   DIS2   DIS3   DIS4  <-states
//------ ------ ------ ------ ------ ------ ------ ------ ------
 {     0, _0100, _0200, _0300, _0400, _0500, _0600, _0700, _0800 }, // (0) TMO <
-events
 {     0,     0, _0201,     0,     0,     0,     0,     0,     0 }, // (1) CON
 {     0,     0, _0202, _0302, _0402, _0502, _0602, _0702,     0 }, // (2) DIS
 { _0003,     0,     0,     0, _0403,     0,     0,     0,     0 }, // (3) SND
 {     0, _0104,     0, _0304,     0,     0, _0604, _0704,     0 }, // (4) RSP
 {     0,     0, _0205,     0,     0,     0,     0,     0,     0 }, // ([illegible]SY
 {     0,     0, _0206,     0,     0,     0,     0,     0,     0 }, // ([illegible]T
 {     0,     0, _0207,     0,     0,     0,     0,     0,     0 }, // ([illegible]NG
 {     0,     0, _0208,     0,     0,     0,     0,     0,     0 }, // (8) NAN
};

/** The State Machine Driver ************************************************/
static void StateMachine (int event)
{
#if TRACE_STATE
        char traceStr[20];
        NWsprintf(traceStr, MSG("-%d%d", 610), DloState, event);
        fputs(traceStr, stdout);
#endif
        if (statetable[event][DloState])
        {
                (*statetable[event][DloState]) ();
        }
}

/*****************************************************************************
*
*       DloSend(BYTE *buffer,
*                       int size,
*                       int timeout)
*
*       Description:
*               This routine is called to send data out the modem. If we[illegible]re
ady
*               connected send it. Otherwise store the message and return. The
*               state machine will send the data after it has connected.
*
*       Input:
*               buffer                                          - data to send
*               size                                                    - size of data
*               timeout                                         - inactivity timeout
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
*****************************************************************************/

int DloSend( LPSTR buffer, int size, int timeout)
{
        int     i;
        LONG *maxBufferSize, *xmitCount;
        BYTE *xmitBuffer;

        if (DloNextConn == DLO_CONN_IDLE)
        {
                /* Assume package delivery connection first */
                DloNextConn = DLO_CONN_PACKAGE;

                if (timeout == DLO_PACKAGE_TIMEOUT)
                {
                        DloPInactivityTimer = DloCfg.pdeliv_inactivity_timer * 1
9;
                }
                else if (timeout == DLO_INET_TIMEOUT ||
                         timeout == DLO_INET_NO_TIMEOUT)
                {
                        DloIInactivityTimer = DloCfg.tinet_inactivity_timer * 19
;
                        DloNextConn = DLO_CONN_INET;
                }
                else if (timeout == DLO_GETKEYS_TIMEOUT)
                {
                        DloPInactivityTimer = 30 * 19;
                }
                else if (timeout > 2)
                {
                        DloPInactivityTimer = timeout * 19;
                }
                else
                {
                        DloPInactivityTimer = 2 * 19;
                }
        }

#if USE_AIO_DEADMAN
        /*
         * Update AIO deadman timer to timeout + 5 seconds
         */

        AIOSetExternalControl(AIOPortHandle, AIO_SET_DEADMAN_TIMER, timeout + 5)
;
#endif

        if (size > DLOBUFSIZE || size < 1)
                return 0;

        if (DloState == DLOS_CONN && DloNextConn == DloConn)
        {
                if (timeout != DLO_INET_NO_TIMEOUT)
                {
                        DloNextConn = DLO_CONN_IDLE;
                        StateMachine(DLOE_SEND);
                }
                UpdateModemLights(1, 0, 1);

                if (DebugFlag)
                {
                        printf(MSG("Sending %d bytes:\n", 485), size);
                        HexAsciiDump(buffer,
                                     (DloConn == DLO_CONN_INET && size > 32) ? 3
2 : size);
                }

                SendAIOData(buffer, size);
                return size;
        }
```

```
        if (DloNextConn == DLO_CONN_PACKAGE)
        {
                maxBufferSize = &DloPMaxBufferSize;
                xmitCount = &DloPXmitCount;
                xmitBuffer = DloPXmitBuffer;
        }
        else
        {
                maxBufferSize = &DloIMaxBufferSize;
                xmitCount = &DloIXmitCount;
                xmitBuffer = DloIXmitBuffer;
        }

        if (size >= *maxBufferSize - *xmitCount)
                return 0;
        for( i = 0; (i < size) && (*xmitCount < *maxBufferSize); i++, (*xmitCoun
t)++)
        {
                if (!*xmitCount)
                        DloStartPacketLifeTimer();
                xmitBuffer[*xmitCount] = buffer[i];
        }

        if (DloConn == DLO_CONN_IDLE)
        {
                StateMachine (DLOE_SEND);
                DloConn = DloNextConn;
                DloNextConn = DLO_CONN_IDLE;
        }
        else if (DloConn == DLO_CONN_INET)
        {
                /* Package waiting for internet. Cause it to timeout quickly */
                DloStartTimer(19);
        }
        return size;
}

/*****************************************************************************
*
*       WriteCommXmitBuffer(void)
*
*       Description:
*               This routine is called after the modem is connected to send the
*               data stored in Dlo?XmitBuffer to the modem.
*
*       Input:
*               nothing
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
*****************************************************************************/

static void WriteCommXmitBuffer( void )
{
        BYTE *xmitBuffer;
        LONG *xmitCount;

        if (DloConn == DLO_CONN_PACKAGE)
        {
                xmitBuffer = DloPXmitBuffer;
                xmitCount = &DloPXmitCount;
        }
        else
        {
                xmitBuffer = DloIXmitBuffer;
                xmitCount = &DloIXmitCount;
        }

        UpdateModemLights(1, 0, 1);

        if (*xmitCount)
        {
                if (DebugFlag)
                {
                        printf(MSG("xmit %d bytes:\n", 486), *xmitCount);
                        HexAsciiDump(xmitBuffer,
                                        (DloConn == DLO_CONN_INET && *xmitCount > 3
2) ? 32 : *xmitCount);
                }

                SendAIOData(xmitBuffer, *xmitCount);
                *xmitCount = 0;
        }
}

/***********************************************************************
*
*       DloStartPacketLifeTimer(void)
*
*       Description:
*               This routine starts the Packet Life timer.
*               This timer is set to the configured packet timer(120 seconds by
*               default) each time data is added via DloSend.
*               DloMain will periodically decrement the timer and clear the
*               data buffer when it reaches zero.
*
*       Input:
*               nothing
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
***********************************************************************/

static void DloStartPacketLifeTimer( void )
{
        DloPacketLifeTimer = (DloCfg.packet_lifetime * 19 > MIN_TIMER_VALUE ?
                DloCfg.packet_lifetime * 19 : MIN_TIMER_VALUE);
}

/***********************************************************************
*
*       DloStopPacketLifeTimer(void)
*
*       Description:
*               The routine is called to stop the packet life timer.
*
*       Input:
*               nothing
*
*       Output:
*               nothing
*
*       Returns:
```

```
*               nothing
*
***********************************************************************/

static void DloStopPacketLifeTimer( void )
{
        DloPacketLifeTimer = 0;
}

/***********************************************************************
*
*       DloStartTimer(LONG ticks)
*
*       Description:
*               This routine is called to set the value of the state machine tim
er.
*               DloMain periodically decrements this timer and set the TIMEOUT
*               event when it hits zero.
*
*       Input:
*               ticks                                                   - Number of tick
s to wait
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
***********************************************************************/

static void DloStartTimer( LONG ticks )
{
        DloTimer = (ticks > MIN_TIMER_VALUE ? ticks : MIN_TIMER_VALUE);
#if TRACE_STATE
 {
        char traceStr[20];
        NWsprintf(traceStr, MSG("*%d", 611), DloTimer);
        fputs(traceStr, stdout);
 }
#endif
}

//static void DloStopTimer( void )
//{
//      DloTimer = 0;
//}

/***********************************************************************
*
*       SendAIOData(BYTE *data,
*                       int length)
*
*       Description:
*               The routine sends the data pointed to by data to the modem.
*
*       Input:
*               data                                            - data to send
*               length                                  - size of data
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
```

```
******************************************************************************/

void SendAIOData(BYTE *data, LONG length)
{
        LONG    count;
        int bufferSpace;
        BYTE *ptr;
        WORD state;
        LONG bytesWritten;

        ptr = data;
        while (ptr < (data + length))
        {
                AIOWriteStatus(AIOPortHandle, &count, &state);
                bufferSpace = AIOWriteBufferSize - count;

                if (length < bufferSpace)
                        bufferSpace = length;

                if (bufferSpace > 0)
                {
                        AIOWriteData(AIOPortHandle, ptr, bufferSpace, &bytesWrit
ten);
                        ptr += bufferSpace;
                }
                else
                        ThreadSwitchWithDelay();
        }
}

/*****************************************************************************
*
*       AIOPortInfo(int portChoice,
*                       int *hardwareType,
*                       int *boardNumer,
*                       int *portNumber)
*
*       Description:
*               The routine returns the AIO informaion of the port passed in
*               portChoice.
*
*       Input:
*               portChoice                              - port to return data ab
out
*               hardwareType                    - where to return hardware type
*               boardNumber                             - where to return board
number
*               portNumber                              - where to return port n
umber
*
*       Output:
*               hardwareType, boardNumber and portNumber filled in if successful
*
*       Returns:
*               0 if successful
*
******************************************************************************/

int AIOPortInfo(int portChoice,
                int *hardwareType,
                int *boardNumber,
                int *portNumber)
{
        int     ccode;
        AIOPORTINFO             portInfo;
        AIOPORTSEARCH   portSearch;
```

```
        char    portOwner[130];

        portInfo.returnLength = sizeof (AIOPORTINFO);

        ccode = AIOGetFirstPortInfo(-1, -1, -1, &portSearch, &portInfo,
                        NULL, NULL, portOwner);

        if (ccode)
                return (-1);

        while (!ccode)
        {
                if (portInfo.portNumber == portChoice)
                {
                        if (portInfo.availability == AIO_AVAILABLE_FOR_ACQUIRE)
                        {
                                *hardwareType = portInfo.hardwareType;
                                *boardNumber  = portInfo.boardNumber;
                                *portNumber   = portInfo.portNumber;
                                return 0;
                        }
                        else
                        {
                                return (-2);
                        }
                }
                ccode = AIOGetNextPortInfo(&portSearch, &portInfo, NULL, NULL,
                                portOwner);
        }
        return -1;
}

/*****************************************************************************
*
*       InitializeAIO(void)
*
*       Description:
*               Initialize AIO. If it didn't initialize, AIOPortHandle will
*               still be negative.
*
*       Input:
*               nothing
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
******************************************************************************/

void InitializeAIO(void)
{
        int     ccode;
        int hardware = AIO_HARDWARE_TYPE_WILDCARD;
        int port = AIO_PORT_NUMBER_WILDCARD;
        int board;
        AIODRIVERLIST dvr;
        dvr.returnLength = sizeof(AIODRIVERLIST);

        if (AIOPortHandle >= 0)
                return;

        while (AIOGetDriverList(hardware, &dvr) == AIO_SUCCESS)
        {
                AIOBOARDLIST brd;
                brd.returnLength = sizeof(AIOBOARDLIST);
                hardware = dvr.driver[0].hardwareType;
                board = AIO_BOARD_NUMBER_WILDCARD;
                while (AIOGetBoardList(hardware, board, &brd) == AIO_SUCCESS)
                {
                        board = brd.board[0].boardNumber;
                        if (strcmp("DPCN_MODEM", brd.board[0].name) == ESUCCESS)
                                goto foundBoard;
                }
        }
        AIOPortHandle = -3;
        return;

foundBoard:
        if (AIOAcquirePortWithRTag(&hardware, &board, &port,
                                &AIOPortHandle, (LONG)asyncIOTag))
        {
                AIOPortHandle = -2;
                return;
        }

        if (AIOSetExternalControl(AIOPortHandle,
                        AIO_EXTERNAL_CONTROL,
                        AIO_EXTCTRL_DTR | AIO_EXTCTRL_RTS))
        {
                AIOReleasePort(AIOPortHandle);
                AIOPortHandle = -1;
                return;
        }

//      if (AIOSetExternalControl(AIOPortHandle,
//                      AIO_BREAK_CONTROL,
//                      AIO_SET_BREAK_ON))
//      {
//              AIOReleasePort(AIOPortHandle);
//              AIOPortHandle = -1;
//              return;
//      }

#if USE_AIO_DEADMAN
        /*
         * Lets set up a default deadman timer of 60 seconds for
         * call setup.
         */

        if (AIOSetExternalControl(AIOPortHandle,
                        AIO_SET_DEADMAN_TIMER,
                        60))
        {
                AIOReleasePort(AIOPortHandle);
                AIOPortHandle = -1;
                return;
        }
#endif

        if ((ccode = AIOConfigurePort(AIOPortHandle,
                        AIOBaudRateDefines[DloCfg.pdeliv_baud_index],
                        AIO_DATA_BITS_8, AIO_STOP_BITS_1,
                        AIO_PARITY_NONE,
                        AIO_SOFTWARE_FLOW_CONTROL_OFF | AIO_HARDWARE_FLOW_CONTRO
L_ON)) )
        {
                if (ccode == AIO_QUALIFIED_SUCCESS)
                {
//                      AIOGetPortConfiguration( AIOPortHandle, &aioPortConfig,
//                                      &aioDrvConfig);
```

```
//                      if (aioPortConfig.bitRate == AIOBaudRateDefines[DloCfg.p
deliv_baud_index]
//                                      && aioPortConfig.dataBits == AIO_DATA_BI
TS_8
//                                      && aioPortConfig.stopBits == AIO_STOP_BI
TS_1
//                                      && aioPortConfig.parityMode == AIO_PARIT
Y_NONE
//                                      && aioPortConfig.flowCtrlMode ==
//                                                      (AIO_SOFTWARE_FLOW_CONTR
OL_OFF | AIO_HARDWARE_FLOW_CONTROL_ON))
//                      {
                                ccode = 0;
//                      }
                }
                if (ccode != 0)
                {
                        AIOReleasePort(AIOPortHandle);
                        AIOPortHandle = -1;
                        return;
                }
        }
        AIOSetWriteBufferSize(AIOPortHandle, DloCfg.async_buffer_size);
        AIOGetWriteBufferSize(AIOPortHandle, &AIOWriteBufferSize);
        DloPMaxBufferSize = (AIOWriteBufferSize < DLOBUFSIZE) ? AIOWriteBufferSi
ze : DLOBUFSIZE;
        DloIMaxBufferSize = (AIOWriteBufferSize < DLOBUFSIZE) ? AIOWriteBufferSi
ze : DLOBUFSIZE;
}

/*****************************************************************************
 *
 *      DloScheduleReceive(void (*callBack)(), int timeout)
 *
 *      Description:
 *              Allow the DLO thread to read a message for you from the modem.
 *              If the DLO is already receiving a message for another process,
 *              an error is returned. Otherwise, the DLO thread waits for the
 *              previous request to be sent out to the modem, and waits for
 *              the reply. The caller must provide a routine to be called
 *              once the message is received.
 *
 *      Input:
 *              callBack                                        - Routine to be
called once message is received
 *              timeout                                 - Number of secon   w
ait before giving up
 *
 *      Output:
 *              nothing
 *
 *      Returns:
 *              0 if the receive has been successfully scheduled
 *
 *****************************************************************************/

int     DloScheduleReceive(void (*callBack)(), int timeout, int callBackType)
{
        if (DloCallBack != 0)
                return(-1);

        AIOFlushBuffers(AIOPortHandle, AIO_FLUSH_READ_BUFFER);
        DloCallBack = callBack;
        DloCallBackTimeout = timeout * 19;
        DloCallBackType = callBackType;
        DloCallBackWait = 1;
        DloCallBackIndex = 0;
        DloCallBackEscape = 0;
        DloCallBackStarted = 0;
        return(0);
}

/**********************************************************************
*
*       ValidPacket(BYTE *buf_to_rx,
*                       int *len_to_rx)
*
*       Description:
*               This routine validates a response from the modem. It checks the
header
*               length, checksum, opcode and status.
*
*       Input:
*               buf_to_rx                               - pointer to the respons
e message
*               len_to_rx                               - pointer to the total l
ength of the message
*
*       Output:
*               len_to_rx                               - changed to the real si
ze of the message
*                                                                       withou
t the header
*
*       Returns:
*               TRUE if packet is valid
*
**********************************************************************/

int     ValidExplicitPacket(BYTE *buf_to_rx, int *len_to_rx)
{
        LroEsPkt_t *msg;
        WORD    frameCrc = 0;
        WORD    slipCrc = 0;
        WORD    frameLen = 0;
        WORD    crcVal = 0xffff;

        msg = (LroEsPkt_t *)buf_to_rx;

        frameLen = (msg->length) & 0x7fff;

        if ((*len_to_rx - sizeof(WORD)) == frameLen)
        {
                CMovB(&buf_to_rx[frameLen], (BYTE *)&frameCrc, sizeof(frameCrc))
;
                slipCrc = calccrc(crcVal, buf_to_rx, frameLen);
                if (slipCrc == frameCrc)
                {
                        return(1);
                }
        }

        return(0);

}

int ValidPacket(BYTE *buf_to_rx, int *len_to_rx)
{
        unsigned short frameCrc=0;      // CRC value contained in the frame
   unsigned short frameLen=0;      // Length value contained in the frame
   unsigned short slipCrc=0;       // CRC returned from calculation
   LroAsyncMsg_t *msg;
```

```
   DacauResponse_t *d_msg;
   int status = FALSE;

   msg = (LroAsyncMsg_t *)buf_to_rx;
   // extract the frame length contained in the first 2 bytes of the frame
   frameLen = msg->header.length;
   frameLen &= 0x7fff;

   if ((*len_to_rx - sizeof(unsigned short)) == frameLen)
   {
        // since slipLen includes CRC
      // extract the crc contained in the last 2 bytes of the frame
      frameCrc = msg->data[frameLen - LRO_ASYNC_HDR_SIZE];
      frameCrc += msg->data[frameLen - LRO_ASYNC_HDR_SIZE + 1] * 256;

      slipCrc = calccrc (INITCRC, (unsigned char *) buf_to_rx, frameLen);
      if (slipCrc == frameCrc)
      {
        d_msg = (DacauResponse_t *)(msg->data);
        if(d_msg->opcode == 1 && d_msg->status == 0)
        {
                *len_to_rx = frameLen - RETURN_POINT;
            status = TRUE;
         }
      }
        }

   return status;
}

/**********************************************************************
*
*       DloCallBackRead(void)
*
*       Description:
*               This routine reads as many bytes as it can from the modem
*               on behalf of the process that scheduled a receive thru
*               DloScheduleReceive(). It's called by the main DLO thread
*               as long as a call back is scheduled(DloCallBack != 0) and
*               the modem has sent the previous request. All Slip specific
*               characters are stripped and all Slip escape characters are
*               converted back to ASCII. If the end of the message is hit,
*               call the processes event routine with the message.
*
*       Input:
*               nothing
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
**********************************************************************/

static void DloCallBackRead()
{
        BYTE    value;

        for(;;)
        {
                if (DloReceive(&value, 1) == 0)
                        break;

                if (DebugFlag)
                        putchar(value);
                if (DloCallBackStarted == 0)
                {
                        if (value == END)
                        {
                                DloCallBackStarted = 1;
                        }
                        continue;
                }

                switch(value)
                {
                        /* if it's an END character then we're done with the pac
ket */
        case END:
                                        /* We're done. */
                                        if (DloCallBackType == EXPLICIT_RECEIVE)
                                        {
                                                if (ValidExplicitPacket(DloCallB
ackBuffer, &DloCallBackIndex))
                                                {
                                                        DloCallBack(DloCallBackB
uffer + LRO_EXPLICIT_HDR_SIZE,
                                                                        DloCallB
ackIndex, 0);
                                                        DloCallBack = 0;
                                                        return;
                                                }
                                                else
                                                {
                                                        DloCallBackIndex = 0;
                                                        DloCallBackStarted = 0;
                                                        DloCallBackEscape = 0;
                                                        break;
                                                }
                                        }
                                        else
                                        {
                                                if (ValidPacket(DloCallBackBuffe
r, &DloCallBackIndex))
                                                {
                                                        DloCallBack(DloCallBackB
uffer + RETURN_POINT + LRO_ASYNC_HDR_SIZE,
                                                                        DloCallB
ackIndex, 0);
                                                        DloCallBack = 0;
                                                        return;
                                                }
                                                else
                                                {
                                                        DloCallBackIndex = 0;
                                                        DloCallBackStarted = 0;
                                                        DloCallBackEscape = 0;
                                                        break;
                                                }
                                        }
                        case ESC:
                                if (DloCallBackEscape == 0)
                                {
                DloCallBackEscape = 1;
                                        break;
                                }
            // here we fall into the default handler and let
            // it store the character for us
                        default:
```

```
                                        if (DloCallBackEscape)
                                        {
                                                switch(value)
                        {
                                case ESC_END:
                                value = END;
                                  break;
                                                        case ESC_ESC:
                                value = ESC;
                                 break;
                                                }
                                        }
                                        DloCallBackEscape = 0;

                                        if(DloCallBackIndex < 4000)
                                                DloCallBackBuffer[DloCallBackIndex++] =
value;
                else
                {
                        /* We're done */
                                                if (DloCallBackType == EXPLICIT_RECEIVE)
                                                {
                                                        DloCallBack(DloCallBackBuffer +
LRO_EXPLICIT_HDR_SIZE,
                                                                        DloCallBackIndex
, 1);
                                                }
                                                else
                                                {
                                                        DloCallBack(DloCallBackBuffer +
RETURN_POINT + LRO_ASYNC_HDR_SIZE,
                                                                        DloCallBackIndex
, 1);
                                                }
                                                DloCallBack = 0;
                                                return;
                        }
                                        break;
                        }
                }
}

LONG DPCNextRegistrationCheck;

/************************************************************************
*
*       DloMain(void *parm)
*
*       Description:
*               Main thread for modem handling.
*               We first initialize the modem thru AIO. We then check the
*               modem to see if we've received anything. If we did, we gather
*               it until we get a carriage return. Then we check against expecte
d
*               responses. If we get a expected response, we set the appropriate
*               event in the state machine. We then check for packet life
*               and state machine timeouts. Otherwise, we sleep.
*
*       Input:
*               parm                                            - ignored
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
```

```
 *
 ***********************************************************************/

void    DloMain(void *parm)
{
        LONG    curticks, delta;
        LONG    count, bytesRead;
        WORD    state;
        BYTE    value;
        int event, eventLen;
        char *pPtrAtBegin, *pPtrAtEnd;
        LONG    extStatus;

        parm = parm;

        DloLastKnownTickCount = GetCurrentTime();

        while(!ExitingFlag)
        {
                delay(1000 / 6);

                count = 0;
                bytesRead = 0;
                if (AIOGetPortStatus(AIOPortHandle,
                                     &count, /* write count */
                                     0, /* write status */
                                     &bytesRead, /* read count */
                                     0, /* read status */
                                     &extStatus,
                                     0) == 0)
                {
                        extStatus &= AIO_EXTSTA_DCD;
                        if (extStatus != DloLastDCD)
                        {
                                if (!extStatus)
                                        StateMachine(DLOE_DISCONN);
                                DloLastDCD = extStatus;
                        }
                }
                UpdateModemLights(count, bytesRead, DloLastDCD);

                if (DloState == DLOS_IDLE)
                {
                        if (DloPXmitCount)
                        {
                                DloConn = DLO_CONN_PACKAGE;
                                StateMachine(DLOE_SEND);
                        }
                        else if (DloIXmitCount)
                        {
                                DloConn = DLO_CONN_INET;
                                StateMachine(DLOE_SEND);
                        }
                }

                curticks = GetCurrentTime();

                if (curticks < DloLastKnownTickCount)
                {
                        delta = curticks + (0xffffffff - DloLastKnownTickCount);
                }
                else
                {
                        delta = curticks - DloLastKnownTickCount;
                }
                DloLastKnownTickCount = curticks;
```

```
                        if (DIOStats && curticks > DPCNextRegistrationCheck)
                        {
                                char buf[16];
                                LONG hw;
                                double key;
                                CustVars* customPtr = (CustVars*)(&DIOStats->CustomVaria
bleCount);
                                LONG rxFreq = customPtr->CustomVariable[1] / 10;

                                DPCSetMaxConnections((LONG*)&key);
                                if (strncmp(DloCfg.base_license, "Helius, Inc.", 8) == 0
) {
                                  DPCNextRegistrationCheck = (LONG)(-1);
                                  goto skipRegistrationCheck;
                                }

                                /* get hardware serial number */
                                *buf = 0;
                                DIOGetSN(buf);
                                hw = strtoul(buf, 0, 10);
                                if (hw == 0) {
                                  ConsolePrintf("\r\nDPCAgent: could not obtain hardware
 serial number of DPC card\n");
                                  goto disableDPCAgent;
                                }

                                /* compute the registration key */
                                if (DebugFlag == 0x98) {
                                        printf("\rRegCheck: %e\n", key);
                                        HexAsciiDump((void*)&key, sizeof(key));
                                }
                                key *= hw + DPC_IP_Address;
                                if (DebugFlag == 0x98) {
                                        printf("\rRegCheck: %e %d %08x\n",
                                               key, hw, DPC_IP_Address);
                                        HexAsciiDump((void*)&key, sizeof(key));
                                }
                                if (key == *(double*)DloCfg.key)
                                        DPCNextRegistrationCheck = curticks + 131072; /*
 120 minutes */
                                else
                                {
                                        ConsolePrintf("\r\nDPCAgent: detected a bad regi
stration key\n");
                                disableDPCAgent:
                                        DPCMaxConnections = 3;
                                        RingTheBell();
                                        if (DPCNextRegistrationCheck) {
                                          DloCfg.gateway_address = 0;
                                          StateMachine(DLOE_TIMEOUT);
                                          DPCNextRegistrationCheck = curticks + 2185; /*
 2 minutes */
                                        }
                                        else
                                          DPCNextRegistrationCheck = curticks + 131072;
/* 120 minutes */
                                }
                                if ((DPCNextRegistrationCheck & 0xfffffff0) == 0xfffffff
0)
                                        DPCNextRegistrationCheck += 17; /* wrap */
                        }

                skipRegistrationCheck:
                        if (AIOPortHandle >= 0)
                        {
```

```
                        for(;;)
                        {
                                if (AIOReadStatus(AIOPortHandle, &count, &state)
 || count == 0)
                                        break;
                                AIOReadData(AIOPortHandle, &value, 1, &bytesRead
);
                                if (DebugFlag && DloState != DLOS_CONN)
                                        putchar(value);

                                if (DloRcvCount < DLOBUFSIZE)
                                {
                                        DloRcvBuffer[DloRcvIndex] = value;
                                        DloRcvIndex++;
                                        if (DloRcvIndex >= DLOBUFSIZE)
                                                DloRcvIndex = 0;
                                        DloRcvBuffer[DloRcvIndex] = 0;
                                        DloRcvCount++;
                                }
                                if (value != '\n' && value != '\r')
                                {
                                        memcpy(DloCommandBuffer, DloCommandBuffe
r + 1, DLOCMDBUFSIZE - 1);
                                        DloCommandBuffer[DLOCMDBUFSIZE - 1] = va
lue;
                                        if (DloCommandIndex < DLOCMDBUFSIZE)
                                                DloCommandIndex++;
                                }
                                if (value != '\r')
                                        continue;

                                pPtrAtBegin = DloCommandBuffer + DLOCMDBUFSIZE -
 DloCommandIndex;
                                eventLen = strlen(DloCompareString[DLOE_CONNECT]
);
                                if (strncmp(pPtrAtBegin, DloCompareString[DLOE_C
ONNECT], eventLen) == 0)
                                {
                                        memcpy(ConnectBaudStr, pPtrAtBegin + eve
ntLen, DloCommandIndex - eventLen);
                                        ConnectBaudStr[DloCommandIndex - eventLe
n + 1] = 0;
                                        DloFlushReceive();
                                        StateMachine(DLOE_CONNECT);
                                }
                                else
                                {
                                        for (event = 0; event < DLOENUM ; event+
+ )
                                        {
                                                if (event == DLOE_CONNECT)
                                                        continue;

                                                eventLen = CStrLen(DloCompareStr
ing[event]);
                                                if (eventLen == 0)
                                                        continue;

                                                pPtrAtEnd = DloCommandBuffer + D
LOCMDBUFSIZE - eventLen;
                                                if (strncmp(pPtrAtEnd, DloCompar
eString[event], eventLen) != 0)
                                                        continue;

                                                DloFlushReceive();
                                                StateMachine(event);
```

```
                                                break;
                                        }
                                }

                                DloCommandIndex = 0;

                        }
                }
                if (delta)
                {
                        /*
                         * Check DLO state machine timer.
                         *
                         */
                        if (DloTimer)
                        {
                                if (delta >= DloTimer)
                                {
                                        DloTimer = 0;
                                        StateMachine(DLOE_TIMEOUT);
                                }
                                else
                                        DloTimer -= delta;
                        }

                        /*
                         * Check Packet lifetime timer.
                         * This timer is initialized when data is added to the
                         * data buffer via DloSend.
                         * If the timer expires, clear the data buffer.
                         */

                        if (DloPacketLifeTimer)
                        {
                                if (delta >= DloPacketLifeTimer)
                                {
                                        DloPacketLifeTimer = 0;
                                        if (DloConn == DLO_CONN_PACKAGE)
                                                DloPXmitCount = 0;
                                        else
                                                DloIXmitCount = 0;
                                }
                                else
                                        DloPacketLifeTimer -= delta;
                        }

                        /*
                         * Check the Receive Call Back timer.
                         * If it times out, call call back routine with timeout
flag.
                         */

                        if (DloCallBack)
                        {
                                if (DloCallBackWait)
                                {
                                        /* Wait until the send request is sent b
efore starting timer */
                                        if (DloEmpty())
                                        {
                                                DloCallBackWait = 0;
                                        }
                                }
                                else
                                {
```

```
                                                        if (delta >= DloCallBackTimeout)
                                                        {
                                                                /* Timeout!!! Call routine with 
timeout flag set */

                                                                DloCallBackWait = 0;
                                                                DloCallBack(DloCallBackBuffer, D
loCallBackIndex, 1);

                                                                DloCallBack = 0;
                                                        }
                                                        else
                                                                DloCallBackTimeout -= delta;

                                                        /* Read any available modem characters f
or call back routine */

                                                        DloCallBackRead();
                                                }
                                        }
                                }
                        }
        if (AIOPortHandle >= 0)
                AIOReleasePort(AIOPortHandle);

        DPCModemPID = 0;
}
```

```
page ,130
;
;**************************************************************\
;
; BEGIN_MANUAL_ENTRY( History, DPC/HISTORY )
;
;       DirecPC Driver Code for NetWare 386.
;       This driver must be loaded after MSM.NLM and ETHERNET.NLM.
;
;       Written by:     DFS
;       Date:           November, 1995
;
;**************************************************************
;
; History Log:
;       version 0.0 - First demo version
;       version 0.2 - (3-15-96) Put SignalQuality and RxFreq into stats table
;       version 0.3 - (4-09-96) Fixed CloseChannel so that DPCAGENT.NLM
;                     can be properly reloaded.
;       version 0.4 - (4-16-96) Added code to send turbo internet packets
;                     up to the ethertsm instead of DPCAGENT
;       version 0.5 - (4-24-96) Adjusted RPacketOffset 2 bytes into
;                     envelope in order to build 14 byte ethernet header
;                     over DPC's 12 byte header to turbo internet packets
;       version 0.6 - (5-16-96) Completion of Phase 2
;                     Debug screen only active if -DEBUG on the command line
;       version 1.00 - (6-11-96)
;                     Added check to end of DriverInit for locked adapter
;       version 1.01 - (6-17-96)
;                     Added Agent Register call to DriverManagement so that
;                     we can call agent when we're being removed, allowing
;                     agent to go into idle state until we're loaded again.
;       version 1.02 - (6-20-96)
;                     Added Freq= to command line.
;       version 1.03 - (6-24-96)
;                     Fixed packet too big bug.
;       version 1.04 - (6-25-96)
;                     Really Fixed packet too big bug.
;       version 1.05 - (6-28-96)
;                     Enhanced dpcagent.nlm
;       version 1.06 - (7-10-96)
;                     Reduced MLIDMaximumSize so Netscape wouldn't overflow
;                     when tunneling enabled.
;       version 1.07 - (7-13-96)
;                     Stopped calling SendComplete from DriverSend
;                     Added DriverTCBComplete to be obtained from IOCTL
;                     Bumped TxFreeCount to 32
;       version 1.09a - (8-28-96)
;                     Reduced max packet size if we couldn't hook bind event.
;       version 1.20 - 2-12-96
;                     Returned ECB allocated by FastProcessGetRCB
;                     Changed call to DataLinkRxFrame to DPCRxFrame
;                     Changed copyright string
;
;
; END_MANUAL_ENTRY
;
;**************************************************************/
;
        name    DPC
        title   DirecPC LAN Driver (HSM Version)
        subttl  -- Structures and Equate Values --
        page

include driver.inc
        subttl  -- DirecPC Specific Equates --
```

```
        page
;
;**************************************************************\
;                                                              *
;       Equates.                                               *
;                                                              *
;**************************************************************/
;
        subttl  -- DirecPC Specific Structures --
        page

MLID_MAJOR_VERSION      equ     01
MLID_MINOR_VERSION      equ     20

TRUE            equ     01
FALSE           equ     00

TIMESTAMP               equ     FALSE
TIMESTAMP_BUFFER_SIZE   equ     (256 * 4)

INT_3           equ     int     3

EVENT_PROTOCOL_BIND             equ     33
EVENT_PROTOCOL_UNBIND   equ     34
;
; Status Register bit masks
;
STAT_BUSY       equ     01              ;/* Adapter is rxing data                  */
STAT_ERROR      equ     02              ;/* One or more errors listed below:       */
                                        ;/* framing,CRC,Abort,Alignment,DES         */
STAT_RX_INT     equ     04h             ;/* Rx interrupt                           */
STAT_NO_RBD     equ     08h             ;/* RBD not ready (E bit wasn't cleared*/
                                        ;/* by software)                           */
STAT_CNTL_INT equ       10h             ;/* Controller Interrupt-RISC handshake*/
STAT_SOUTPUT    equ     20h             ;/* From CNTL Spare Output (0x800)         */
;
; Control Register bit masks
;
CNTL_CPU_EN     equ     01h             ;/* RISC CPU Enable                        */
CNTL_RX_EN      equ     02h             ;/* Rx Enable                              */
CNTL_INT_EN     equ     04h             ;/* Interrupt Enable                       */
CNTL_CHAN_ATT equ       08h             ;/* Another handshake with RISC CPU        */
CNTL_SNGL_INT equ       10h             ;/* Bits 4-7: # of buffers per Rx Int      */
CNTL_15_INT     equ     F0h             ;/* (from 1 to 15)                         */
CNTL_AUTO_INC equ       100h            ;/* Auto Inc Message RAM Pointer           */
CNTL_MRESET     equ     200h            ;/* Soft Reset and Clear All Registe
CNTL_FORCEINT equ       400h            ;/* 'Cause Interrupt' bit                  */
CNTL_SOUTPUT    equ     800h            ;/* Spare Output to STAT Bit 5 (0x20)      */
CNTL_IRQ3       equ     0000h           ;/* Bits 13-15: IRQ selection              */
CNTL_IRQ4       equ     2000h
CNTL_IRQ5       equ     4000h
CNTL_IRQ9       equ     6000h
CNTL_IRQ10      equ     8000h
CNTL_IRQ11      equ     0A000h
CNTL_IRQ12      equ     0C000h
CNTL_IRQ15      equ     0E000h
;
; Synthesizer chip types
;
INVALID_TUNER                   equ     0ffh    ;/* not supported */
SHARP                           equ     00h
ALPS                            equ     08h
PANASONIC                       equ     20h
SHARP_CUSTOM                    equ     28h
;
; Define State Numbers(informational only)
```

```
;
INIT                        equ     0
SYNTH_PRGM                  equ     1
ACQ_PD_DELAY                equ     2
ACQ_PD                      equ     3
ENABLE_BTR                  equ     4
START_SEARCH_FOR_FEC        equ     5
CHECK_FOR_FEC_LOCK          equ     6
SET_OTHER_MODE              equ     7
TRACKING                    equ     8
POINTING_ACQ                equ     9
POINTING_TRACKING           equ     10
HALT                        equ     11
;
; New Stuff DBS
; demod command definitions
;
ACQUIRE_MODE                equ     0       ;/** start a new acquisition        **/
HALT_MODE                   equ     2       ;/** Do nothing                     **/
BUSY_MODE                   equ     3       ;/** Trying to acquire              **/
POINTING_MODE               equ     4       ;/** Special test mode              **/


DEFAULT_RX_FREQ equu     1330

BIT_OFF                     equ     00h
;
; BtrControlAddr bits - write register 0
;
FREQ_PWR_MASK               equ     0E0h
PHASE_PWR_MASK              equ     07h
BTR_SENSE_MASK              equ     08h
BTR_ERR_ENA_MASK            equ     10h
FREQ_PWR_OFFSET             equ     20h
PHASE_PWR_OFFSET            equ     01h
;
; AfcControlAddr bits - write register 1
;
SWP_ENA_MASK                equ     01h
SWEEP_DIR_SENSE_MASK        equ     08h
AFC_SENSE_MASK              equ     10h
EXT_INT_MASK                equ     20h
BPSK_MASK                   equ     40h
ROM_ENA_MASK                equ     80h
SQF_PEAK_EN_MASK            equ     02h
;
; BitDetControlAddr bits - write register 2
;
SOFT_THRS_MASK              equ     1Fh
DECODER_INFC_SEL_MASK       equ     80h
VIT_SEQ_MASK                equ     40h
SOFT_THRS_OFFSET            equ     01h
;
; AgcFirControlAddr bits - write register 3
;
AGC_REF_MASK                equ     1Fh
AGC_SENSE_MASK              equ     40h
FIR_BYPASS_MASK             equ     80h
SWAP_IQ_MASK                equ     20h
AGC_REF_OFFSET              equ     01h
;
; CrlkControlAddr bits - write register B
;
CRLK_DET_PWR_MASK           equ     07h
CRLK_FC_MASK                equ     38h
CRLK_GAIN_MASK              equ     C0h
```

```
CRLK_DET_PWR_OFFSET         equ     01h
CRLK_FC_OFFSET              equ     08h
CRLK_GAIN_OFFSET            equ     40h
;
; SynthSerControlAddr :bits - write register C
;
SDATA_MASK                  equ     01h
SCLK_MASK                   equ     02h
SENA_MASK                   equ     04h
MODE_MASK                   equ     08h
CRL_ACC_ENABLE              equ     10h
DEPUNC_BYPASS_MASK          equ     20h
RESET_FEC_ACQ_MASK          equ     40h
RESET_BTR_ACC_MASK          equ     80h
;
; DaAdOffsetControlAddr bits - write register E
;
I_CHANNEL_OFFSET            equ     01h
CRL_ERROR_OFFSET            equ     08h
BTR_ERROR_OFFSET            equ     40h
;
;
;
SYNTH_LOCK_MASK             equ     01h
CRL_LOCK_MASK               equ     02h
SWEEPING_MASK               equ     04h
DIR_MASK                    equ     08h
FEC_LOCK_MASK               equ     10h
TUNER_TYPE_MASK             equ     20h
TUNER_TYPE_2_MASK           equ     08h
VENDOR_ID_MASK              equ     0f0h
;
;/* Frequency offsets */
;
PLUS_OFFSET                 equ     40
ZERO_OFFSET                 equ     0
MINUS_OFFSET                equ     -40
OFFSET_THRESHOLD            equ     1152     ;/* for SHARP type tuners */
FREQ_BASE                   equ     9011
K_TRACK                     equ     1736
K_REACQ                     equ     29622
NOM_COUNT_TRACK             equ     48761
NOM_COUNT_REACQ             equ     19432
SYNTH_CHANNEL_SIZE          equ     3645
SYNTH_RATIO                 equ     64
SYNTH_CHANNELS_PER_STEP equ     40
SYNTH_FIRST_CHANNEL         equ     3788

BPSK                        equ     BPSK_MASK OR ROM_ENA_MASK
;
;/** Possible Viterbi modes **/
;
LOWRATE                     equ     0
HIGHRATE                    equ     1
;
;/** demod status states **/
;
UNLOCKED                    equ     0
LOCKED                      equ     1
;
; Configuration equates
;
RBD_BASE_ADDR               equ     0a000h
ADAP_RBD_NUM                equ     128
RBD_BUFFER_SIZE             equ     1024
LOCAL_BUF_NUM               equ     400
```

```
;
; RBD status bits.
;
FRAMING_ERR             equu    0001h           ; Framing error
CRC_ERR                 equ     0002h           ; CRC error
ABORT                   equ     0004h           ; Abort
ALIGN_ERR               equ     0008h           ; Alignment error
DES_ERR                 equ     0010h           ; DES Error
SOF_BIT                 equ     0020h           ; Start of Frame(not used)
EOF_BIT                 equ     0040h           ; End of Frame
OVERRUN_ERR             equ     0080h           ; Frame Overrun bit
EMPTY                   equ     8000h           ; if reset, buffer may be used
STATUS_ERROR            equ     FRAMING_ERR OR CRC_ERR OR ABORT OR ALIGN_ERR OR
DES_ERR OR OVERRUN_ERR

DebugMessage    macro   mask, message

        local   DebugMessageExit

        test    DebugMask, mask
        je      DebugMessageExit

        push    eax
        push    ecx
        push    edx

        push    offset message
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]

        pop     edx
        pop     ecx
        pop     eax
DebugMessageExit:
                endm

DebugMessage1   macro   mask, message, parm0

        local   DebugMessageExit

        test    DebugMask, mask
        je      DebugMessageExit

        push    eax
        push    ecx
        push    edx

        push    parm0
        push    offset message
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (3 * 4)]

        pop     edx
        pop     ecx
        pop     eax
DebugMessageExit:
                endm

DebugMessage2   macro   mask, message, parm0, parm1

        local   DebugMessageExit

        test    DebugMask, mask
        je      DebugMessageExit
```

```
        push    eax
        push    ecx
        push    edx

        push    parm1
        push    parm0
        push    offset message
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (4 * 4)]

        pop     edx
        pop     ecx
        pop     eax
DebugMessageExit:
                endm

DebugMessage3   macro   mask, message, parm0, parm1, parm2

        local   DebugMessageExit

        test    DebugMask, mask
        je      DebugMessageExit

        push    eax
        push    ecx
        push    edx

        push    parm2
        push    parm1
        push    parm0
        push    offset message
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (5 * 4)]

        pop     edx
        pop     ecx
        pop     eax
DebugMessageExit:
                endm

DebugMessage4   macro   mask, message, parm0, parm1, parm2, parm3

        local   DebugMessageExit

        test    DebugMask, mask
        je      DebugMessageExit

        push    eax
        push    ecx
        push    edx

        push    parm3
        push    parm2
        push    parm1
        push    parm0
        push    offset message
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (6 * 4)]

        pop     edx
        pop     ecx
        pop     eax
```

```
DebugMessageExit:
                endm


DebugMessage5   macro   mask, message, parm0, parm1, parm2, parm3, parm4

        local   DebugMessageExit

        test    DebugMask, mask
        je      DebugMessageExit

        push    eax
        push    ecx
        push    edx

        push    parm4
        push    parm3
        push    parm2
        push    parm1
        push    parm0
        push    offset message
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (7 * 4)]

        pop     edx
        pop     ecx
        pop     eax
DebugMessageExit:
                endm

DebugMessage6   macro   mask, message, parm0, parm1, parm2, parm3, parm4, parm5

        local   DebugMessageExit

        test    DebugMask, mask
        je      DebugMessageExit

        push    eax
        push    ecx
        push    edx

        xor     eax, eax
        mov     al, parm5
        push    eax
        mov     al, parm4
        push    eax
        mov     al, parm3
        push    eax
        mov     al, parm2
        push    eax
        mov     al, parm1
        push    eax
        mov     al, parm0
        push    eax
        push    offset message
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (8 * 4)]

        pop     edx
        pop     ecx
        pop     eax
DebugMessageExit:
                endm
```

```
SLOW            macro
        push    eax
        in      al, 61h
        in      al, 61h
        in      al, 61h
        pop     eax

                endm

BufferStruct    struc
        BufPtr          dd      0               ; Pointer to buffer
        DataSize        dd      0               ; Size of buffer
BufferStruct    ends

LAST_EOF                equ     80000000h       ; or with DataSize

MAX_APPL_RBD            equ     350
MAX_CHAN                equ     10
RBD_NOT_USED            equ     -1
MAX_CONF_ADDR           equ     8
MAX_ADDR                equ     16

RX_CNTL struc

        RxChannel       dd      0               ; Channel ID
        RxESR           dd      0               ; ESR to call when a packet
                                                ;  is received.
RX_CNTL ends

RX_FLAG_STATS_ONLY      equ     1

ChannelConfig   struc

        CfgChannel      dd      0               ; Channel ID
        CfgESR          dd      0               ; max # of buffers used
        CfgNumAddresses dd      0               ; Number of filter addrs
                                                ; that is to follow
        CfgAddress      db      6 dup (0)       ; list of addrs
        CfgGroupKey     db      8 dup (0)       ; Group keys
        CfgElementKey   db      8 dup (0)       ; Element keys

ChannelConfig   ends

FilterStruct    struc

        FilterAddress           db      6 dup (0)       ; 48 bit address
                                db      2 dup (0)       ; Align next field
        FilterChannel           dd      0               ; Channel ID
        FilterCmdblkIndex       dd      0               ; RISC SW cmd blk index
        FilterTotalCount        dd      0               ; # of frames rxed
        FilterSeqCount          dd      0               ; # of out-of-seq frames
        FilterSeqNum            dd      0               ; next seq expected

FilterStruct    ends

EblkStruct      struc

        EblkCmd                 dw      0               ;
        EblkPortID              dw      0               ;
        EblkNotUsed             dw      0               ;
        EblkAddress             db      6 dup (0)       ;
        EblkGroupKey            db      8 dup (0)       ;
        EblkElemKey             db      8 dup (0)       ;

EblkStruct      ends
;
```

```
;***************************************************************\
;                                                               *
; DirecPC Structures.                                           *
;                                                               *
;***************************************************************/
;
;
; MIPS Code Statistics
;
; NOTE: Only third, fourth, and fifth items are currently used
StatsBlk        struc
    rxenables           dd      0       ; number of times the RSW has gotten a r
x
                                        ; enable cmd.
    rxdisables          dd      0       ; number of rx disable commands.
    framesAccepted      dd      0       ; total number of frames accepted.
    noFilterMatch       dd      0       ; number of frames rejected due to no fi
lter match.
    zeroAddrFrames      dd      0       ; number of frames rejected due to an
                                        ; address of all zeroes or RBD shortage
    disabledRejed       dd      0       ; number of frames which had to be rejec
ted while zeroed.
    fullBufsRejec       dd      0       ; number of frames rejected because the
                                        ; adapter was close to running out of bu
ffers (RBDS).
StatsBlk        ends

;
;*********************************************************\
;                                                         *
; MulticastTableStructure.                                *
;                                                         *
;*********************************************************/
;
MulticastTableStructure struc

        MulticastAddress        db      6 dup (0)
        EntryUsed               dw      0

MulticastTableStructure ends
;
;*********************************************************\
;                                                         *
; Start of the Adapter Data Space structure for DirecPC.  *
;                                                         *
;*********************************************************/
;
DriverAdapterDataSpace  struc

FirstTimeInit           dd      1       ; Start out in driver init.

IORxData                dd      0       ; Rx Data port = base + 0
IOAutoInc               dd      0       ; Auto Inc port = base + 2
IOStatus                dd      0       ; Status port = base + 4
IOControl               dd      0       ; Control port = base + 6
IOMsgRamPtr             dd      0       ; Msg Ram Ptr = base + 8
IOMsgRam                dd      0       ; Msg Ram = base + 10
IORbdBufLen             dd      0       ; RDB buf Len = base + 12
IORbdNum                dd      0       ; RDB Num = base + 14
IOBtrControlAddr        dd      0       ; BTR Control Addr = base + 16
IOAfcControlAddr        dd      0       ; AFC Control Addr = base + 18
IOBitDetControlAddr     dd      0       ; Bit Det Control Addr = base + 20
IOAgcFirControlAddr     dd      0       ; Agc Fir Control Addr = base + 22
IOCrlkThrLowAddr        dd      0       ; Crlk Thr Low Addr = base + 24
IOCthAddr               dd      0       ; Cth Addr = base + 26
IOGateCountHighAddr     dd      0       ; Gate Count High Addr = base + 28
```

```
IOCountNomLowAddr       dd      0       ; Count Nom Low Addr = base + 30
IOCountNomHighAddr      dd      0       ; Count Nom High Addr = base + 32
IOCountDeltaAddr        dd      0       ; Count Delta Addr = base + 34
IOSweepRateAddr         dd      0       ; Sweep Rate Addr = base + 36
IOCrlkControlAddr       dd      0       ; Crlk Control Addr = base + 38
IOSynthSerControlAddr   dd      0       ; Synth Ser Control Addr = base + 40
IOSpareIOControlAddr    dd      0       ; Spare IO Control Addr = base + 42
IODaAdOffsetControlAddr dd      0       ; DA Offset Control Addr = base + 44
IOUnitIDAddr            dd      0       ; Unit ID Addr = base + 46

TunerTypeFound          dd      INVALID_TUNER
ReacqGateCount          dd      ?
ReacqDeltaCount         dd      ?
NomCountSearch          dd      ?
SqfCheckPoints          dd      ?
SqfCheckStepSize        dd      ?
SqfDeltaCount           dd      ?
GLDrift                 dd      ?

PicMask                 dd      ?
PicUnMask               dd      ?
PicAddress              dd      ?

CurrentState            dd      HALT
ViterbiMode             dd      0
ViterbiOnly             dd      FALSE
DemodCommand            dd      HALT_MODE
SearchLoc               dd      1
Drift                   dd      0
GLOffset                dd      0
TrackingMode            dd      FALSE
ChannelNumber           dd      0
NextState               dd      0
ModulationScheme        dd      BPSK
TuneCount               dd      0
BestTuneCount           dd      0
RateCount               dd      0
PointingFlag            dd      0
DemodStatus             dd      0
FecStatus               dd      0
NextStepCount           dd      1
SearchLocFound          dd      FALSE
MaxSqf                  dd      0
SqfAvg                  dd      0
SqfWait                 dd      0

TimerTag                dd      0
EventTag                dd      0
ISRTag                  dd      0
ProtocolBindID  dd      0
ProtocolUnbindID        dd      0

T1Count                 dd      0
T2Count                 dd      0
MaxCount                dd      0

IntStatus               dd      0
CurrentAdapterRBD       dd      0
CurrentECB              dd      0

RxControl               db      (size RX_CNTL *  MAX_CHAN) dup (0)

Filter                  db      (size FilterStruct * MAX_ADDR) dup (0)

Eblk                    db      size EblkStruct dup (0)
```

```
EblkBusyFlags           db      32 dup (0)

BufferCount             dd      0
WatchBufferCount        dd      0
WatchOldRejected        dd      0
;;LocalMipsStats        db      (size StatsBlk) dup (0)
AgentSendRoutine        dd      0
AgentRemoveRoutine      dd      0

GotInterrupt            dd      0
IOEnableValue           dd      0
;
;***********************************************************\
;                                                           *
;       Error Counters.                                     *
;                                                           *
;***********************************************************/
;
StatisticsVersion       db      03, 00
GenericVariableCount    dw      (RxAbortFrameAlignment + 4 - TotalTxPacketCount)
 / 4
NotSupportedMask0       dd      11111011110010000000000000000011b

        TotalTxPacketCount      dd      0               ; 1 - (Used by MSM)
        TotalRxPacketCount      dd      0               ; 1 - (Used by MSM)
        NoECBAvailableCount     dd      0               ; 1 - (Used by MSM)
        PacketTxTooBigCount     dd      0               ; 1 - not used
        PacketTxTooSmallCount   dd      0               ; 1 - not used
        PacketRxOverflowCount   dd      0               ; 0 - Used by driver
        PacketRxTooBigCount     dd      0               ; 1 - not used
        PacketRxTooSmallCount   dd      0               ; 1 - not used
        PacketTxMiscErrorCount  dd      0               ; 1 - not used
        PacketRxMiscErrorCount  dd      0               ; 1 - not used
        RetryTxCount            dd      0               ; 0 - Used by driver
        ChecksumErrorCount      dd      0               ; 0 - Used by driver
        HardwareRxMismatchCount dd      0               ; 1 - (Used by MSM)
        TotalTxOKByteCountLow   dd      0               ; 0 - Used by MSM
        TotalTxOKByteCountHigh  dd      0               ; 0 - Used by MSM
        TotalRxOKByteCountLow   dd      0               ; 0 - Used by MSM
        TotalRxOKByteCountHigh  dd      0               ; 0 - Used by MSM
        TotalGroupAddrTxCount   dd      0               ; 0 - Used by MSM
        TotalGroupAddrRxCount   dd      0               ; 0 - Used by MSM
        AdapterResetCount       dd      0               ; 0 - Used by driver
        AdapterOprTimeStamp     dd      0               ; 0 - Used by MSM
        QDepth                  dd      0               ; 0 - Used by MSM

        TxOKSingleCollision     dd      0               ; 0 - Used by driver
        TxOKMultipleCollisions  dd      0               ; 0 - Used by driver
        TxOKButDeferred         dd      0               ; 0 - not used
        TxAbortLateCollision    dd      0               ; 0 - Used by driver
        TxAbortExcessCollisions dd      0               ; 0 - Used by driver
        TxAbortCarrierSense     dd      0               ; 0 - Used by driver
        TxAbortExDeferral       dd      0               ; 0 - not used
        RxAbortFrameAlignment   dd      0               ; 0 - Used by driver


CustomVariableCount             dw      (MipsFullBufsRejec + 4 - SignalQuality)
/ 4

        SignalQuality                   dd      0       ; Signal Quality
 from adapter
        RxFreq                          dd      0       ; Recieve Freque
ncy

        LargestRx               dd      0
        NumberLargeRx           dd      0
```

```
        AveLargeRx                      dd      0

        MipsRxEnables                           dd      0       ; number of time
s the RSW has gotten a rx
                                        ;  enable cmd.
 MipsRxdisables                         dd      0       ; number of rx disable c
ommands.
 MipsFramesAccepted     dd      0       ; total number of frames accepted.
 MipsNoFilterMatch              dd      0       ; number of frames rejected due
to no filter match.
 MipsZeroAddrFrames     dd      0       ; number of frames rejected due to an
                                ;  address of all zeroes or RBD shortage
 MipsDisabledRejed              dd      0       ; number of frames which had to
be rejected while zeroed.
 MipsFullBufsRejec              dd      0       ; number of frames rejected beca
use the
                                ;  adapter was close to running out of buffers (RBDS
).


VectorToTheStrings      dd      offset DiagnosticsStrings

TotalLargeRx            dd      0

AlignDEndVA             db      ((4 - offset (AlignDEndVA and 3)) and 3) + 4 dup
 (?)    ; Align 4 for MOVSD

DriverAdapterDataSpace  ends

IORbdBase               equ     IOStatus
IOTuningLowAddr         equ     IOCountNomLowAddr
IOTuningHighAddr        equ     IOCountNomHighAddr
IORelSqfAddr            equ     IOCountDeltaAddr
IOMaxSqfAddr            equ     IOSweepRateAddr
IOStatusAddr            equ     IODaAdOffsetControlAddr

        subttl  -- OSDATA Data Segment --
        page
        assume  cs: OSCODE, ds: OSDATA, es: OSDATA, ss: OSDATA
;
;*******************************************************************\
;                                                                   *
; The following variables are common to the entire driver.          *
;                                                                   *
;*******************************************************************/
;
OSDATA  segment rw public 'DATA'

HSMSPEC                 db      'HSM_SPEC_VERSION: 3.2',0
;
;***********************************************************\
;                                                           *
; Statistic Diagnostic Strings.                             *
;                                                           *
;***********************************************************/
;
DiagnosticsStrings      dw      (EndOfStrings-DiagnosticsStrings)

        db      'Signal Quality',0
        db      'Rx Frequency',0

        db      'Largest Received IP Frame', 0
        db      'Number of Receive IP Frames over 400', 0
        db      'Ave size of Receive IP Frames over 400', 0

        db      'Mips Rx enable commands', 0
        db      'Mips Rx disable commands', 0
        db      'Mips frames accepted', 0
        db      'Mips frames rejected from no filter match', 0
        db      'Mips frames rejected from zero address', 0
        db      'Mips frames rejected from adapter disabled', 0
        db      'Mips frames rejected from low buffer pool', 0

        db      0,0

EndOfStrings    equ     $
;
;*********************************************************************\
;                                                                     *
; Driver Parameter Block to pass to MSM.                              *
;                                                                     *
;*********************************************************************/
;
        align   4
DriverParameterBlock            label   dword
DriverParameterSize             dd      DriverParameterBlockSize
DriverStackPointer              dd      0
DriverModuleHandle              dd      0
DriverBoardPointer              dd      0
DriverAdapterPointer            dd      0
DriverConfigTemplatePtr         dd      DriverConfigTemplate
DriverFirmwareSize              dd      0
DriverFirmwareBuffer            dd      0
DriverNumKeywords               dd      2
DriverKeywordText               dd      DPCKeywordText
DriverKeywordTextLen            dd      DPCTextLen
DriverProcessKeywordTab dd      DPCProcessKeywordTab
DriverAdapterDataSpaceSize      dd      SIZE DriverAdapterDataSpace
DriverAdapterTemplate           dd      DriverAdapterDataSpaceTemplate
DriverStatisticsTable           dd      StatisticsVersion
DriverEndOfChainFlag            dd      0
DriverSendWantsECBs             dd      0
DriverMaxMulticast              dd      -1
DriverNeedsBelow16Meg           dd      0
DriverAESPtr                    dd      0
DriverCallBackPtr               dd      offset DriverCallBack
DriverISRPtr                    dd      offset DriverISR
DriverMulticastChangePtr        dd      offset DriverMulticastChange
DriverPollPtr                   dd      0
DriverResetPtr                  dd      offset DriverReset
DriverSendPtr                   dd      offset DriverSend
DriverShutdownPtr               dd      offset DriverShutdown
DriverTxTimeoutPtr              dd      0
DriverPromiscuousChangePtr      dd      offset DriverPromiscuousChange
DriverStatisticsChangePtr       dd      offset RefreshMipsStats
DriverRxLookAheadChangePtr      dd      0
DriverManagementPtr             dd      offset DriverManagement
DriverEnableInterruptPtr        dd      offset DriverEnableInterrupt
DriverDisableInterruptPtr       dd      offset DriverDisableInterrupt

DriverParameterBlockSize        equ     $ - DriverParameterBlock
;
;*********************************************************************\
;                                                                     *
; Driver Management Dispatch Jump Table.                              *
;                                                                     *
;*********************************************************************/
;
ManagementJumpTable     label   dword
        dd      offset GetMipsStats
        dd      offset OpenChannel
        dd      offset CloseChannel
```

```
        dd      offset GetSN
        dd      offset SignText
        dd      offset AddAddress
        dd      offset DeleteAddress
        dd      offset RegisterAgentSendRoutine
        dd      offset BadParametersExit
        dd      offset RegisterAgent
        dd      offset ReturnTCBCompleteRoutine
LastManagementFunction  equ     (($ - ManagementJumpTable) / 4) - 1
;
;*********************************************************************\
;                                                                     *
; Copy of Virtual Adapter Data area to be copied at                   *
; initialization.                                                     *
;                                                                     *
;*********************************************************************/
;
DriverAdapterDataSpaceTemplate  DriverAdapterDataSpace  <>

DriverConfigTemplate    db      'HardwareDriverMLID        '    ; [ebx].MLI[illegible]G_
Signature
        db      01                              ; [ebx].MLIDCFG_MajorVersion
        db      12                              ; [ebx].MLIDCFG_MinorVersion
        db      6 dup (0ffh)                    ; [ebx].MLIDNodeAddress
        dw      0010010001001001b               ; [ebx].MLIDModeFlags
        dw      0000                            ; [ebx].MLIDBoardNumber
        dw      0000                            ; [ebx].MLIDBoardInstance
        dd      00000000                        ; [ebx].MLIDMaximumSize
        dd      00000000                        ; [ebx].MLIDMaxRecvSize
        dd      00000000                        ; [ebx].MLIDRecvSize
        dd      00000000                        ; [ebx].MLIDCardName
        dd      DriverNICShortName              ; [ebx].MLIDshortName
        dd      00000000                        ; [ebx].MLIDFrameType
        dw      0000                            ; [ebx].MLIDReserved0
        dw      0000                            ; [ebx].MLIDFrameID
        dw      0001                            ; [ebx].MLIDTransportTime
        dd      000000000                       ; [ebx].MLIDRouteHandler
        dw      10                              ; [ebx].MLIDLineSpeed
        dw      0000                            ; [ebx].MLIDLookAheadSize
        db      8 dup (00h)                     ; [ebx].MLIDReserved
        db      MLID_MAJOR_VERSION                      ; [ebx].MLIDMajo
rVersion
        db      MLID_MINOR_VERSION                      ; [ebx].MLIDMino
rVersion
        dw      0010b                           ; [ebx].MLIDFlags
        dw      0010                            ; [ebx].MLIDSendRetries
        dd      00000000                        ; [ebx].MLIDLink
        dw      0000                            ; [ebx].MLIDSharingFlags
        dw      0000                            ; [ebx].MLIDSlot
        dw      02c0h, 40h, 0, 0                ; [ebx].MLIDIOPortsAndLengths
        dd      00000000                        ; [ebx].MLIDMemoryDecode0
        dw      0000                            ; [ebx].MLIDLength0
        dd      00000000                        ; [ebx].MLIDMemoryDecode1
        dw      0000                            ; [ebx].MLIDLength1
        db      03, 0FFh                        ; [ebx].MLIDInterrupt
        db      0FFh, 0FFh                      ; [ebx].MLIDDMAUsage
        dd      00000000                        ; [ebx].MLIDResourceTag
        dd      00000000                        ; [ebx].MLIDConfiguration
        dd      00000000                        ; [ebx].MLIDCommandString
        db      18 dup (0)                      ; [ebx].MLIDLogicalName
        dd      00000000                        ; [ebx].MLIDLinearMemory0
        dd      00000000                        ; [ebx].MLIDLinearMemory1
        dw      0000                            ; [ebx].MLIDChannelNumber
        db      6 dup (0)                       ; [ebx].MLIDIOReserved

Message         DriverNICShortName,     'DPC'
```

```
;
;***********************************************************\
;                                                          *
; Parameters required by ParseDriverParameters.            *
;                                                          *
;***********************************************************/
DebugText               db      '-DEBUG'        ; Break into debugger keyword.
DebugTextLen            equ     $-DebugText OR T_STRING
FreqText                db      'FREQ'          ; Set RxFreq
FreqTextLen             equ     $-FreqText OR T_NUMBER
                        dd      0               ; min
                        dd      10000           ; max
                        dd      0
                        dd      0
                        dd      0

DPCKeywordText  dd      DebugText       ; First Keyword.
                dd      FreqText        ; Second Keyword.
DPCTextLen      dd      DebugTextLen    ; First Keywords length.
                dd      FreqTextLen     ; Second Keywords length.


DPCProcessKeywordTab    dd      DebugRoutine    ; First Keyword routine.
                        dd      FreqRoutine     ; Second Keyword routine.

GlobalRxFreq            dd      DEFAULT_RX_FREQ
;
IOPort0Data             dd      12, 100h, 140h, 180h, 1c0h
                        dd      200h, 240h, 280h, 2c0h
                        dd      300h, 340h, 380h, 3c0h
Interrupt0Data          dd      8, 3, 4, 5, 9, 10, 11, 12, 15

AdapterOptions  AdapterOptionDefinitionStructure        <,IOPort0Data,,,,,,,,Int
errupt0Data>

SearchTbl               dd      MINUS_OFFSET
                        dd      ZERO_OFFSET
                        dd      PLUS_OFFSET

StateTbl                dd      offset InitState
                        dd      offset SynthPrgmState
                        dd      offset AcqPDDelayState
                        dd      offset AcqPDState
                        dd      offset EnableBTRState
                        dd      offset StartSearchForFECState
                        dd      offset CheckforFECLockState
                        dd      offset SetOtherModeState
                        dd      offset TrackingState
                        dd      offset PointingAcquisitionState
                        dd      offset PointingTrackingState
                        dd      offset HaltState

MajorVer                dd      0
MinorVer                dd      0
Year                    dd      0
Month                   dd      0
Day                     dd      0
ScreenRTag              dd      0
DPCScreen               dd      0

OurAdapterDataSpace     dd      0
;
;***********************************************************\
;                                                          *
; Mipx code.                                               *
; Contains array MipsCode and dword MipsCodeSize.          *
```

```
;                                                          *
;***********************************************************/
;
include mips.dat
;
;
;***********************************************************\
;                                                          *
; Message equates.                                         *
;                                                          *
;***********************************************************/
;
CR                              equ     13
LF                              equ     10
;
;***********************************************************\
;                                                          *
; Run-time error message strings.                          *
; DPC error messages.                                      *
;                                                          *
;***********************************************************/
;
TransmitTimeoutMessage  db      66, 00, 'The cable might be disconnected on the
board.', CR, LF, 0
DMANotCompleteError     db      200, 00, "The board's DMA did not complete the w
rite.", CR, LF, 0
;
;***********************************************************\
;                                                          *
; InitNIC error message strings.                           *
;                                                          *
;***********************************************************/
;
ErrorAllocatingRTagMessage      db      72, 00, 'A resource tag is unavailable.'
, CR, LF, 0
NICNotInSlotMessage     db      50, 00, 'The board cannot be found.', CR, LF, 0
PortFailMessage         db      54, 00, 'The board did not respond to the initia
lization command.', CR, LF, 0
NICIn8BitSlotMessage    db      223, 00, 'The board must be placed in a 16-bit s
lot.', CR, LF, 0
NICIsNE1000Message      db      224, 00, 'This board is configured as an NE1000.
', CR, LF, 0
NICBadConfiguration     db      80, 00, 'The board could not be configured.', CR
, LF, 0
BadISRMsg               db      207, 00, 'Unable to set ISR for test ISR.',
LF, 0
NoInterruptMsg  db      208, 00, 'Interrupt is non functional. Try using another
 interrupt.', CR, LF, 0
ErrorAllocatingMemoryMessage    db      73, 00, 'Unable to allocate memory.', CR
, LF, 0

MsgIOSetFailed          db      200, 00, 'Unable to set the spare output. Choose
 another base I/O port.', CR, LF, 0
MsgIOClearFailed        db      201, 00, 'Unable to clear the spare output. Choo
se another base I/O port.', CR, LF, 0
MsgBadRAM               db      51, 00, 'Board RAM failed the memory test.', CR,
 LF, 0

LockedAdapterMsg        db      202, 00, 'Unable to start the adapter. Power dow
n system and reboot.', CR, LF, 0

TimerDesc               db      "DPC Timer", 0
EventRTagMessage        db      "DPC Events", 0
InterruptRTagMessage    db      "DPC Test ISR", 0

LastDebugMessage        dd      ?
```

```
LastSignalStrength      dd      0

SignalStrengthMsg       db      "Signal Strength = %d", CR, LF, 0
HaltStateMsg            db      "Halt State entered", CR, LF,0
PointingTrackStateMsg   db      "Pointing Tracking State entered", CR, LF, 0
PointingAcqStateMsg     db      "Pointing Acquisition State entered", CR, LF, 0
TrackingStateMsg        db      "Tracking State entered", CR, LF, 0
SetOtherModeStateMsg    db      "Set Other Mode state entered", CR, LF, 0
CheckforFECLockStateMsg db      "Check For FEC Lock state entered", CR, LF, 0
StartSearchForFECMsg    db      "Start Search For FEC state entered", CR, LF, 0
EnableBTRMsg            db      "Enable BTR state entered", CR, LF, 0
AcqPDMsg                db      "Acq PD state entered", CR, LF, 0
AcqPDDelayMsg           db      "Acq PD Delay state entered", CR, LF, 0
SynthPrgmMsg            db      "Synth Prgm state entered", CR, LF, 0
InitStateTrackMsg       db      "Init state entered - Tracking Enabled", CR, LF,
 0
InitStateNoTrackMsg     db      "Init state entered - Tracking Disabled", CR, LF
, 0
ChannelNumberMsg        db      "Channel Number = %d", CR, LF, 0
SharpTunerMsg           db      "Sharp Tuner was detected", CR, LF, 0
PanasonicTunerMsg       db      "Panasonic Tuner was detected", CR, LF, 0
SharpCustomTunerMsg     db      "Sharp Custom Tuner was detected", CR, LF, 0
ScreenResourceName      db      "DPC Screen", 0
ScreenName              db      "DirecPC Trace", 0
DebugRBDReceived        db      "Received Buffer %d", CR, LF, 0
DebugRBDSize            db      "Buffer Size = %d", CR, LF, 0
DebugScreenHeader       db      "%S v%2u.%2u, %2u/%02u/%u", CR, LF, LF, 0
AdapterRBDMsg           db      " ISR:Adapter RBD  set to %d", CR, LF, 0
FilterAddrMsg           db      " ISR:Filter Address = %2.2x %2.2x %2.2x %2.2x %
2.2x %2.2x", CR, LF, 0
FilterRBDsMaxed         db      " ISR:Error: We maxed Filter RBDs, max = %d, we
wanted %d", CR, LF, 0
FilterRBDLen            db      " ISR:Error: Length mismatch. Total Reported siz
e = %d, headers = %d", CR, LF, 0
FilterCopy              db      " ISR:Copying from rbd element %d to filter elem
ent %d", CR, LF, 0
FilterNone              db      " ISR:Error, no Filter for %02.2x %02.2x %02.2x
%02.2x %02.2x %02.2x", CR, LF, 0
ISREnterMsg             db      "ISR:Entering ISR", CR, LF, 0
ISRExitMsg              db      "ISR:Exiting ISR - Adapter RBD set to %d", CR, L
F, 0
FramingErrMsg           db      "FRAMING_ERR", 0
AbortMsg                db      "  ABORT_ERR", 0
AlignErrMsg             db      "  ALIGN_ERR", 0
OverrunErrMsg           db      "  OVERRUN_ERR", 0
DESErrMsg               db      "  DES_ERR", 0
CRCErrMsg               db      "  CRC_ERR", 0
NoECBMsg                db      " ISR Error: No ECB buffers."
ReturnMsg               db      CR, LF, 0
AddAddrMsg              db      "IOCTL: Adding Address %x", CR, LF, 0
DebugInitOK             db      "DirecPC Adapter Initialized successfully at "
                        db      "I/O port %xh interupt %d", CR, LF, 0
IPName                  db      2, "IP", 0
NLMName                 db      128 dup (0)
;
; If command line contains "-DEBUG", DebugMask will be set to DEBUG_MASK
;
DEBUG_ISR               equ     0001h
DEBUG_ISR_ALL           equ     0002h
DEBUG_IOCTL             equ     0004h

DEBUG_MASK              equ     DEBUG_ISR OR DEBUG_ISR_ALL

DebugMask               dd      0

        align   4
```

```
SEND_STACK_SIZE equu    4000h

SendStackEnd            dd      SEND_STACK_SIZE dup (0)
SendStackBegin  dd      0
;;OldSendStack          dd      0

        public DPCRxFrame
DPCRxFrame      dd      0

if TIMESTAMP
        public  DPCTB
DPCTB   db      TIMESTAMP_BUFFER_SIZE dup (0)
timestamp_begin dd      0
timestamp_end           dd      0
timestamp_index dd      0

DPCDead         dd      0
endif

OSDATA  ends
        subttl  -- DriverChangeMulticast --
        page
OSCODE  segment er public 'CODE'

DebugRoutine    proc

        mov     DebugMask, DEBUG_MASK
        ret

DebugRoutine    endp

FreqRoutine     proc

        mov     GlobalRxFreq, eax
        ret

FreqRoutine     endp
;
;*******************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverManagement, DPC/API/MANAGE )
;
; Name:         DriverManagement
;
; Description:  This routine will handle any DirecPC specific
;               commands called by a remote nlm.
;
;
; On Entry:     EAX     N/A
;               EBX     @ FrameDataSpace
;               ECX     N/A
;               EDX     N/A
;               EBP     @ Adapter Data Space
;               ESI     @ ECB
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     0 if successful
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Destroyed
;               EDI     Destroyed
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by the ethernet media module.
;               It can be called at process or interrupt time.
;
; See Also:     ETHERTSM\EtherTSMAddMulticastAddress
;               ETHERTSM\EtherTSMDeleteMulticastAddress
;               ETHERTSM\EtherTSMUpdateMulticast
;
; END_MANUAL_ENTRY
;
;*********************************************************************/
;
DriverManagement        proc
;
;*************************************************************\
;                                                             *
; First reset Multicast Address Registers.                    *
;                                                             *
;*************************************************************/
;
        cmp     dword ptr [esi].RProtocolID+0, 'TCRD'   ; ID+0 == 'DRCT'?
        jne     BadParametersExit                       ; Jump if not
        cmp     word ptr [esi].RProtocolID+4, 'CP'      ; ID+4 == 'PC'?
        jne     BadParametersExit                       ; Jump if not

        movzx   ecx, [esi].RLogicalID                   ; ECX = Function
        cmp     ecx, LastManagementFunction             ; Supported?
        ja      BadParametersExit
        jmp     ManagementJumpTable[ecx * 4]

BadParametersExit:
        mov     eax, BadParameters
        ret

DriverManagement        endp
        subttl  -- SignText --
        page
;
;*********************************************************************\
;
; BEGIN_MANUAL_ENTRY( SignText, DPC/API/SIGNTXT )
;
; Name:         SignText
;
; Description:  This routine is called by DriverManagement to
;               sign the text passed in.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     ECB
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
```

```
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverManagement.
;               It is called at process time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*********************************************************************/
;
        public  SignText
SignText        proc

        mov     esi, [esi].RPacketOffset

        mov     edi, [esi+0]
        or      edi, edi
        je      SignTextError

        cmp     dword ptr [esi+8], 0
        je      SignTextError

        cli
        mov     edx, [ebp].IOMsgRamPtr
        mov     eax, 18800h / 2
        out     dx, ax
        mov     ecx, [esi+4]
        shr     ecx, 1
        mov     edx, [ebp].IOMsgRam
        inc     ecx
SendStringLoop:
        mov     ax, [edi]
        out     dx, ax
        add     edi, 2
        dec     ecx
        jne     SendStringLoop
        sti

        cli
        mov     edx, [ebp].IOMsgRamPtr
        mov     eax, (18100h + (14 * size EblkStruct) + 4) / 2
        out     dx, ax

        mov     edx, [ebp].IOMsgRam
        mov     eax, [esi + 4]
        out     dx, ax

        mov     edx, [ebp].IOMsgRamPtr
        mov     eax, (18100h + (14 * size EblkStruct)) / 2
        out     dx, ax

        mov     edx, [ebp].IOMsgRam
        mov     eax, 10h
        out     dx, ax

        sti

        mov     ecx, 10
        mov     edi, 10h
```

```
WaitForCommandDone:

        push    ecx
        mov     eax, [ebp].TimerTag
        push    eax
        push    2
        call    DelayMyself
        add     esp, (2 * 4)
        pop     ecx

        cli
        mov     edx, [ebp].IOMsgRamPtr
        mov     eax, (18100h + (14 * size EblkStruct)) / 2
        out     dx, ax

        mov     edx, [ebp].IOMsgRam
        in      ax, dx
        sti
        cmp     ax, 0eeh
        je      SignTextError

        cmp     ax, 11h
        je      ReadyToGetSignature

        dec     ecx
        jne     WaitForCommandDone

ReadyToGetSignature:
        mov     edi, [esi+8]
        cli

        mov     edx, [ebp].IOMsgRamPtr
        mov     eax, 18790h / 2
        out     dx, ax

        mov     ecx, 4
        mov     edx, [ebp].IOMsgRam
GetSignatureLoop:
        in      ax, dx
        mov     [edi], ax
        add     edi, 2
        dec     ecx
        jne     GetSignatureLoop

        sti
        mov     dword ptr [esi+4], 8
        xor     eax, eax
        ret

SignTextError:
        mov     eax, -1
        ret

SignText        endp
        subttl  -- GetSN --
        page
;
;*****************************************************************\
;
; BEGIN_MANUAL_ENTRY( GetSN, DPC/API/GETSN )
;
; Name:         GetSN
;
; Description:  This routine is called by DriverManagement to
;               return the adapters serial number.
;
```

```
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     ECB
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverManagement.
;               It is called at process time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*****************************************************************/
;
        public  GetSN
GetSN   proc

        mov     esi, [esi].RPacketOffset

        cli
        mov     edx, [ebp].IOMsgRamPtr
        mov     eax, 18760h / 2
        out     dx, ax

        mov     ecx, 3
        mov     edx, [ebp].IOMsgRam
GetSNLoop:
        in      ax, dx
        mov     [esi], ax
        add     esi, 2
        dec     ecx
        jne     GetSNLoop

        sti
        mov     [esi], cl
        ret


GetSN   endp
        subttl  -- CloseChannel --
        page
;
;*****************************************************************\
;
; BEGIN_MANUAL_ENTRY( CloseChannel, DPC/API/CLSCHAN )
;
; Name:         CloseChannel
;
; Description:  This routine is called by DriverManagement to
;               close the specified channel.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     ECB
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverManagement.
;               It is called at process time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*********************************************************************/
;
        public  CloseChannel
CloseChannel    proc

        mov     esi, [esi].RPacketOffset        ; ESI -> config structure
        mov     esi, [esi]                      ; ESI = channel

        cmp     esi, MAX_CHAN
        ja      CloseChannelError
        lea     edi, [ebp].RxControl[esi*8]
        cmp     [edi].RxChannel, RBD_NOT_USED
        je      CloseChannelError

        mov     [edi].RxChannel, RBD_NOT_USED
        mov     [edi].RxESR, 0

        mov     ecx, MAX_ADDR
        lea     edi, [ebp].Filter
CloseChannelCloseFilterLoop:
        cmp     [edi].FilterChannel, esi
        jne     CloseChannelCloseFilterNext

        mov     [edi].FilterChannel, RBD_NOT_USED
        mov     eax, [edi].FilterCmdblkIndex
        mov     [ebp].EblkBusyFlags[eax], 0

        push    ecx
        mov     ecx, size EblkStruct
        mul     ecx
        mov     edx, [ebp].IOMsgRamPtr
        add     eax, 18100h
        shr     eax, 1
        push    eax
        add     eax, 3
```

```
        out     dx, ax

        mov     edx, [ebp].IOMsgRam
        xor     eax, eax
        out     dx, ax
        out     dx, ax
        out     dx, ax

        pop     eax
        pop     ecx
        mov     edx, [ebp].IOMsgRamPtr
        out     dx, ax
        mov     edx, [ebp].IOMsgRam
        mov     eax, 1
        out     dx, ax

CloseChannelCloseFilterNext:
        add     edi, size FilterStruct
        dec     ecx
        jne     CloseChannelCloseFilterLoop

        xor     eax, eax
        ret

CloseChannelError:
        mov     eax, -1
        ret

CloseChannel    endp
        subttl  -- AddAddress --
        page
;
;*********************************************************************\
;
; BEGIN_MANUAL_ENTRY( AddAddress, DPC/API/ADDADRS )
;
; Name:         AddAddress
;
; Description:  This routine is called by DriverManagement to
;               add the address passed in.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     ECB
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverManagement.
;               It is called at process time.
;
```

```
; See Also:
;
; END_MANUAL_ENTRY
;
;**********************************************************************/
;
        public  AddAddress
AddAddress      proc

        mov     esi, [esi].RPacketOffset
        mov     eax, [esi].CfgChannel
        cmp     eax, MAX_CHAN
        ja      OpenChannelError

        lea     edi, [ebp].RxControl[eax*8]
        cmp     [edi].RxChannel, RBD_NOT_USED
        je      OpenChannelError

        ;
        ; Fall thru to AddAddr
        ;

AddAddress      endp
        subttl  -- AddAddr --
        page
;
;**********************************************************************\
;
; BEGIN_MANUAL_ENTRY( AddAddr, DPC/API/ADDADDR )
;
; Name:         AddAddr
;
; Description:  This routine is called by AddAddress and OpenChannel
;               to add the address to the adapter.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     ECB
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by AddAddress and OpenChannel.
;               It is called at process time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;**********************************************************************/
;
```

```
        public  AddAddr
AddAddr proc


        mov     eax, dword ptr [esi].CfgAddress
        DebugMessage1   DEBUG_IOCTL, AddAddrMsg, eax

        mov     ecx, [esi].CfgNumAddresses                      ; ECX = Number O
f Addrs
AddAddrLoop:

;
; First make sure this address is not a duplicate
;
        lea     edi, [ebp].Filter
        mov     edx, MAX_ADDR
AddAddrDuplicateLoop:
        cmp     [edi].FilterChannel, RBD_NOT_USED
        je      AddAddrDuplicateNext
        mov     eax, dword ptr [edi].FilterAddress
        cmp     eax, dword ptr [esi].CfgAddress
        jne     AddAddrDuplicateNext
        mov     ax, word ptr [edi].FilterAddress+4
        cmp     ax, word ptr [esi].CfgAddress+4
        jne     AddAddrDuplicateNext

        mov     eax, -1
        ret

AddAddrDuplicateNext:
        add     edi, size FilterStruct
        dec     edx
        jne     AddAddrDuplicateLoop
;
; Find an empty slot in the filter table
;
        lea     edi, [ebp].Filter
        xor     edx, edx
AddAddrFindEmptyLoop:
        cmp     [edi].FilterChannel, RBD_NOT_USED
        je      AddAddrFindEmptyFound
        add     edi, size FilterStruct
        inc     edx
        cmp     edx, MAX_ADDR
        jb      AddAddrFindEmptyLoop

        mov     eax, -1
        ret

AddAddrFindEmptyFound:
        mov     eax, [esi].CfgChannel
        mov     [edi].FilterChannel, eax
        mov     eax, dword ptr [esi].CfgAddress
        mov     dword ptr [edi].FilterAddress, eax
        mov     ax, word ptr [esi].CfgAddress+4
        mov     word ptr [edi].FilterAddress+4, ax
        mov     [edi].FilterTotalCount, 0
        mov     [edi].FilterSeqCount, 0
        mov     [edi].FilterSeqNum, 0

        mov     edx, 16
        mov     eax, 31
        test    [esi].CfgAddress+0, 02h
        jnz     AddAddrFindEblkLoop
        mov     edx, 2
        mov     eax, 13
```

```
AddAddrFindEblkLoop:
        cmp     [ebp].EblkBusyFlags[edx], 0
        je      AddAddrFindEblkFound
        inc     edx
        cmp     edx, eax
        jbe     AddAddrFindEblkLoop

        mov     eax, -1
        ret

AddAddrFindEblkFound:
        mov     [ebp].EblkBusyFlags[edx], 1
        mov     [edi].FilterCmdblkIndex, edx

        mov     eax, [edi].FilterChannel
        lea     edi, [ebp].Eblk
        mov     [edi].EblkCmd, 0
        mov     [edi].EblkPortID, ax
        mov     ax, word ptr [esi].CfgAddress
        xchg    ah, al
        mov     word ptr [edi].EblkAddress, ax
        mov     ax, word ptr [esi].CfgAddress+2
        xchg    ah, al
        mov     word ptr [edi].EblkAddress+2, ax

        cmp     edx, 16
        jae     AddAddrIsBypass
        mov     eax, dword ptr [esi].CfgGroupKey
        mov     dword ptr [edi].EblkGroupKey, eax
        mov     eax, dword ptr [esi].CfgGroupKey+4
        mov     dword ptr [edi].EblkGroupKey+4, eax
        mov     eax, dword ptr [esi].CfgElementKey
        mov     dword ptr [edi].EblkElemKey, eax
        mov     eax, dword ptr [esi].CfgElementKey+4
        mov     dword ptr [edi].EblkElemKey+4, eax

AddAddrIsBypass:
        pushfd
        cli
        push    ecx
        mov     eax, edx
        mov     ecx, size EblkStruct
        mul     ecx
        mov     edx, [ebp].IOMsgRamPtr
        add     eax, 18100h
        shr     eax, 1
        push    eax
        push    esi
        out     dx, ax
        mov     edx, [ebp].IOMsgRam
        mov     ecx, size EblkStruct / 2
        mov     esi, edi
        cld
AddAddrCopyEblk:
        lodsw
        out     dx, ax
        dec     ecx
        jne     AddAddrCopyEblk

        pop     esi
        pop     eax
        pop     ecx
        mov     edx, [ebp].IOMsgRamPtr
        out     dx, ax
        mov     edx, [ebp].IOMsgRam
        mov     eax, 1
```

```
        out     dx, ax
        popfd

        dec     ecx
        jne     AddAddrLoop

        xor     eax, eax
        ret

AddAddr endp
        subttl  -- DeleteAddress --
        page
;
;*********************************************************************\
;
; BEGIN_MANUAL_ENTRY( DeleteAddress, DPC/API/DELADDR )
;
; Name:         DeleteAddress
;
; Description:  This routine is called by DriverManagement to
;               delete the address passed in.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     ECB
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverManagement.
;               It is called at process time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*********************************************************************/
;
        public  DeleteAddress
DeleteAddress   proc

        mov     esi, [esi].RPacketOffset        ; ESI -> config structure
        mov     eax, [esi].CfgChannel
        cmp     eax, MAX_CHAN                   ; over the max?
        ja      DeleteAddressError              ; jump if it is.

        lea     edi, [ebp].RxControl[eax*8]
        cmp     [edi].RxChannel, RBD_NOT_USED
        je      DeleteAddressError

        mov     ecx, [esi].CfgNumAddresses
        cmp     ecx, MAX_CONF_ADDR
        ja      DeleteAddressError

        lea     ebx, [esi].CfgAddress
DeleteAddressLoop:

        mov     ch, MAX_ADDR
        lea     edi, [ebp].Filter
DeleteAddressFilterLoop:
        mov     eax, dword ptr [ebx+0]
        cmp     eax, dword ptr [edi].FilterAddress+0
        jne     DeleteAddressNextFilter
        mov     ax, word ptr [ebx+4]
        cmp     ax, word ptr [edi].FilterAddress+4
        jne     DeleteAddressNextFilter

        mov     eax, [esi].CfgChannel
        cmp     eax, [edi].FilterChannel
        jne     DeleteAddressNextFilter

        mov     [edi].FilterChannel, RBD_NOT_USED
        mov     eax, [edi].FilterCmdblkIndex
        mov     [ebp].EblkBusyFlags[eax], 0

        pushfd
        cli
        push    ecx
        mov     ecx, size EblkStruct
        mul     ecx
        mov     edx, [ebp].IOMsgRamPtr
        add     eax, 18100h
        shr     eax, 1
        add     eax, 3
        push    eax
        out     dx, ax
        mov     edx, [ebp].IOMsgRam
        xor     eax, eax
        out     dx, ax
        out     dx, ax
        out     dx, ax

        pop     eax
        pop     ecx
        sub     eax, 3
        mov     edx, [ebp].IOMsgRamPtr
        out     dx, ax
        mov     edx, [ebp].IOMsgRam
        mov     eax, 1
        out     dx, ax
        popfd

DeleteAddressNextFilter:
        add     edi, size FilterStruct
        dec     ch
        jne     DeleteAddressFilterLoop

DeleteAddressNext:
        add     ebx, 6
        dec     cl
        jne     DeleteAddressLoop

        xor     eax, eax
        ret

DeleteAddressError:
        mov     eax, -1
```

```
        ret

DeleteAddress   endp
        subttl  -- RegisterAgentSendRoutine --
        page
;
;*****************************************************************************\
;
; BEGIN_MANUAL_ENTRY( RegisterAgentSendRoutine, DPC/API/DELADDR )
;
; Name:          RegisterAgentSendRoutine
;
; Description:   This routine is called by DriverManagement to
;                register a Slip Send routine. The first dword in the first
;                ECB fragment points to the Slip Send routine to use. To
;                deregister the send routine, set the dword to a NULL.
;
; On Entry:      EAX     N/A
;                EBX     Frame Data Space
;                ECX     N/A
;                EDX     N/A
;                EBP     Adapter Data Space
;                ESI     ECB
;                EDI     N/A
;
;                Note:   Interrupts are in any state.
;
; On Return:     EAX     Destroyed
;                EBX     Preserved
;                ECX     Destroyed
;                EDX     Destroyed
;                EBP     Preserved
;                ESI     Preserved
;                EDI     Preserved
;
;                Flags:
;
;                Note:   Interrupts preserved.
;
; Remarks:       This routine is called by DriverManagement.
;                It is called at process time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*****************************************************************************/
;
RegisterAgentSendRoutine proc

        mov     esi, [esi].RPacketOffset                ; ESI -> config structur
e
        mov     eax, [esi]
; EAX -> Slip Send Routine Address
        mov     [ebp].AgentSendRoutine, eax     ; Save it for later
        xor     eax, eax
        ret

RegisterAgentSendRoutine        endp
        subttl  -- RegisterAgent --
        page
;
;*****************************************************************************\
;
; BEGIN_MANUAL_ENTRY( RegisterAgent, DPC/API/REGAG )
;
```

```
; Name:           RegisterAgent
;
; Description:    This routine is called by DriverManagement to
;                 register package delivery/internet agent. The first dword
;                 in the first ECB fragment points to the Remove routine
;                 that we must call before we are removed.
;
; On Entry:       EAX     N/A
;                 EBX     Frame Data Space
;                 ECX     N/A
;                 EDX     N/A
;                 EBP     Adapter Data Space
;                 ESI     ECB
;                 EDI     N/A
;
;                 Note:   Interrupts are in any state.
;
; On Return:      EAX     Destroyed
;                 EBX     Preserved
;                 ECX     Destroyed
;                 EDX     Destroyed
;                 EBP     Preserved
;                 ESI     Preserved
;                 EDI     Preserved
;
;                 Flags:
;
;                 Note:   Interrupts preserved.
;
; Remarks:        This routine is called by DriverManagement.
;                 It is called at process time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*******************************************************************/
;
RegisterAgent proc

        mov     esi, [esi].RPacketOffset                ; ESI -> config structur
e
        mov     eax, [esi]
; EAX -> Slip Send Routine Address
        mov     [ebp].AgentRemoveRoutine, eax   ; Save it for later
        xor     eax, eax
        ret

RegisterAgent endp
        subttl  -- ReturnTCBCompleteRoutine --
        page
;
;*******************************************************************\
;
; BEGIN_MANUAL_ENTRY( ReturnTCBCompleteRoutine, DPC/API/RETADS )
;
; Name: ReturnTCBCompleteRoutine
;
; Description:    This routine is called by DriverManagement to
;                 return the TCB Complete routine. The first dword
;                 in the first ECB fragment points to a LONG which we will
;                 return with a pointer to the TCB complete routine.
;
; On Entry:       EAX     N/A
;                 EBX     Frame Data Space
;                 ECX     N/A
```

```
;                 EDX     N/A
;                 EBP     Adapter Data Space
;                 ESI     ECB
;                 EDI     N/A
;
;                 Note:   Interrupts are in any state.
;
; On Return:      EAX     Destroyed
;                 EBX     Preserved
;                 ECX     Destroyed
;                 EDX     Destroyed
;                 EBP     Preserved
;                 ESI     Preserved
;                 EDI     Preserved
;
;                 Flags:
;
;                 Note:   Interrupts preserved.
;
; Remarks:        This routine is called by DriverManagement.
;                 It is called at process time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*******************************************************************/
;
if TIMESTAMP
        extrn   GetHighResolutionTimer: near
        extrn   HighResolutionTimer: dword
        public  DPCTimestamp
DPCTimestamp    proc

        CPush
        call    GetHighResolutionTimer

        and     eax, 00ffffffh
        mov     edx, timestamp_index
        mov     ecx, [esp + Parm0]
        shl     ecx, 24
        or      eax, ecx
        mov     [edx], eax

        add     edx, 4
        cmp     edx, timestamp_end
        jae     short DPCTimestampBegin
DPCTimestampExit:
        mov     timestamp_index, edx
        mov     dword ptr [edx], '$$$$'
        CPop
        ret

DPCTimestampBegin:
        mov     edx, timestamp_begin
        jmp     DPCTimestampExit

DPCTimestamp    endp
endif

        subttl -- DriverTCBComplete --
        page

DriverTCBComplete       proc

        CPush
        mov     esi, [esp + Parm0]              ; esi -> TCB
        mov     ebp, OurAdapterDataSpace        ; ebp -> Adapter Data Space
        inc     [ebp].MSMTxFreeCount            ; Add to send resources
        call    EtherTSMFastSendComplete        ; Give it back

        test    [ebp].MSMStatusFlags, SHUTDOWN  ; Don't get next send
        jne     GetNextSendExit                 ;  if we're shutting down.

GetNextSendLoop:
        test    [ebp].MSMStatusFlags, TXQUEUED  ; Any ECB's waiting.
        jnz     short GetNextSendGetIt          ; Jump if there is.

GetNextSendExit:
        CPop
        ret

GetNextSendGetIt:
        call    EtherTSMGetNextSend             ; Get next send if any.
        jne     short GetNextSendExit   ; Jump if no send
        call    DriverSend                      ; Send it.
        jmp     GetNextSendLoop         ; See if we have any more

DriverTCBComplete       endp


ReturnTCBCompleteRoutine proc

        mov     esi, [esi].RPacketOffset                ; ESI -> fragment
        mov     dword ptr [esi], offset DriverTCBComplete
        xor     eax, eax
        ret

ReturnTCBCompleteRoutine endp
        subttl  -- OpenChannel --
        page
;
;*******************************************************************\
;
; BEGIN_MANUAL_ENTRY( OpenChannel, DPC/API/OPENCHAN )
;
; Name:         OpenChannel
;
; Description:  This routine is called by DriverManagement to
;               open a channel on the adapter to receive packets
;               from.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     ECB
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
```

```
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverManagement.
;               It is called at process time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*******************************************************************/
;
        public  OpenChannel
OpenChannel     proc

        mov     esi, [esi].RPacketOffset        ; ESI -> config structure
;
; Don't open if we don't have signal lock.
;
        cmp     [ebp].SignalQuality, 200
        jb      OpenChannelError
;
; Find an empty RxControl entry
;
        xor     ecx, ecx                        ; ECX = index
        lea     edi, [ebp].RxControl            ; EDI -> Rx Control Structures
FindEmptyRBDLoop:
        cmp     [edi].RxChannel, RBD_NOT_USED   ; empty?
        jne     FindEmptyRBDNext                ; Jump if not
;
; Found one. Initialize it before adding addresses.
;
        mov     [edi].RxChannel, ecx            ; Save channel number
        mov     [esi].CfgChannel, ecx
        mov     eax, [esi].CfgESR
        mov     [edi].RxESR, eax

        push    edi
        call    AddAddr
        pop     edi
        or      eax, eax
        jne     OpenChannelDidntAdd
        ret

OpenChannelDidntAdd:
        mov     [edi].RxChannel, RBD_NOT_USED
        mov     [edi].RxESR, 0
        mov     eax, -1
        ret

FindEmptyRBDNext:
        inc     ecx
        add     edi, size RX_CNTL
        cmp     ecx, MAX_CHAN
        jb      FindEmptyRBDLoop

OpenChannelError:
        mov     eax, -1
        ret

OpenChannel     endp
        subttl  -- RefreshMipsStats --
        page
;
;*******************************************************************\
;
; BEGIN_MANUAL_ENTRY( RefreshMipsStats, DPC/API/RFSHMIPS )
```

```
;
; Name:         RefreshMipsStats
;
; Description:  This routine is called by to read the Mips stats
;               from the adapter and to store them into the Local
;               Mips Stats structure.
;
; On Entry:     EAX     N/A
;               EBX     N/A
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Destroyed
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by GetMipsStats and
;               DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*********************************************************************/
;
RefreshMipsStats        proc

        pushfd
        cli
        mov     edx, [ebp].IOMsgRamPtr
        mov     eax, 18700h / 2
        out     dx, ax

        mov     edx, [ebp].IOMsgRam
        lea     edi, [ebp].MipsRxEnables                ; LocalMipsStats
        mov     ecx, (size StatsBlk) / 2
        cld
GetMipsStatsLoop:
        in      ax, dx
        stosw
        loop    GetMipsStatsLoop

        popfd
        xor     eax, eax
        ret

RefreshMipsStats        endp
        subttl  -- GetMipsStats --
        page
;
;*********************************************************************\
;
```

```
; BEGIN_MANUAL_ENTRY( GetMipsStats, DPC/API/GETMIPS )
;
; Name:         GetMipsStats
;
; Description:  This routine is called by DriverManagement to
;               return the Mips statistics.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     ECB
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverManagement.
;               It is called at process time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*********************************************************************/
;
GetMipsStats    proc

        push    esi
        call    RefreshMipsStats
        pop     esi

        mov     edi, [esi].RPacketOffset
        lea     esi, [ebp].MipsRxEnables                ; LocalMipsStats
        mov     ecx, (size StatsBlk) / 4
        cld
        rep     movsd

        xor     eax, eax
        ret

GetMipsStats    endp
;
;*********************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverMulticastChange, DPC/API/MULTI )
;
; Name:         DriverMulticastChange
;
; Description:  This routine will modify the NIC's multicast registers to
;               enable it to receive the multicast addresses listed in
;               the multicast table. Each entry in the multicast table is as
;               follows:
;
```

```
;               bytes 0-5 = Multicast Address.
;               bytes 6-7 = Entry used(Non zero if used).
;
; On Entry:     EAX     N/A
;               EBX     N/A
;               ECX     # of Entries in Table( 0 if empty )
;               EDX     N/A
;               EBP     @ Adapter Data Space
;               ESI     @ Multicast Table
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Destroyed
;               EDI     Destroyed
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by the ethernet media module.
;               It can be called at process or interrupt time.
;
; See Also:     ETHERTSM\EtherTSMAddMulticastAddress
;               ETHERTSM\EtherTSMDeleteMulticastAddress
;               ETHERTSM\EtherTSMUpdateMulticast
;
; END_MANUAL_ENTRY
;
;*************************************************************************/
;
DriverMulticastChange   proc
;
;***************************************************************\
;                                                               *
; First reset Multicast Address Registers.                      *
;                                                               *
;***************************************************************/
;

        ret

DriverMulticastChange   endp
        subttl  -- DriverPromiscuousChange --
        page
;
;*************************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverPromiscuousChange, DPC/API/PROMISCU )
;
; Name:         DriverPromiscuousChange
;
; Description:  This routine will enable/disable the Promiscuous Mode.
;
; On Entry:     EAX     N/A
;               EBX     N/A
;               ECX     0 to disable the Promiscuous mode
;               EDX     N/A
;               EBP     @ Adapter Data Space
;               ESI     @ Multicast Table
;               EDI     N/A
;
```

```
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Destroyed
;               EDI     Destroyed
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by the ethernet media module.
;               It can be called at process or interrupt time.
;
; See Also:     ETHERTSM\EtherTSMPromiscuousChange
;
; END_MANUAL_ENTRY
;
;*************************************************************************/
;
DriverPromiscuousChange proc

        ret

DriverPromiscuousChange endp
        subttl  -- CalculateDriftDelta --
        page
;
;*************************************************************************\
;
; BEGIN_MANUAL_ENTRY( CalculateDriftDelta, DPC/API/CALCDD )
;
; Name:         CalculateDriftDelta
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by InitState.
;               It can be called at process or interrupt time.
;
; See Also:
;
```

```
; END_MANUAL_ENTRY
;
;*****************************************************************/
;
        public  CalculateDriftDelta
CalculateDriftDelta     proc


        mov     edi, [ebp].Drift
        cmp     edi, NOM_COUNT_TRACK
        jbe     DriftBelowNOM

        lea     eax, [edi - NOM_COUNT_TRACK]
        xor     edx, edx
        mov     ecx, 210
        div     ecx
        mov     [ebp].GLDrift, eax

        mov     ecx, 210
        mul     ecx
        shr     eax, 4
        mov     edi, eax
        mov     eax, [ebp].Drift
        sub     eax, NOM_COUNT_TRACK
        sub     eax, edi

        mov     edi, 0ffffh
        sub     edi, [ebp].GLDrift
        inc     edi
        mov     [ebp].GLDrift, edi

        ret

DriftBelowNOM:
        mov     eax, NOM_COUNT_TRACK
        sub     eax, [ebp].Drift
        xor     edx, edx
        mov     ecx, 210
        div     ecx
        mov     [ebp].GLDrift, eax

        mov     ecx, 210
        mul     ecx
        shr     eax, 4
        mov     edi, NOM_COUNT_TRACK
        sub     edi, [ebp].Drift
        sub     edi, eax

        mov     eax, 0ffffh
        sub     eax, edi
        inc     eax
        ret

CalculateDriftDelta     endp
        subttl  -- Step --
        page
;
;*****************************************************************\
;
; BEGIN_MANUAL_ENTRY( Step, DPC/API/STEP )
;
; Name: /       Step
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
```

```
;                       EBX     Frame Data Space
;                       ECX     N/A
;                       EDX     N/A
;                       EBP     Adapter Data Space
;                       ESI     N/A
;                       EDI     N/A
;
;                       Note:   Interrupts are in any state.
;
; On Return:            EAX     Destroyed
;                       EBX     Preserved
;                       ECX     Destroyed
;                       EDX     Destroyed
;                       EBP     Preserved
;                       ESI     Preserved
;                       EDI     Preserved
;
;                       Flags:
;
;                       Note:   Interrupts preserved.
;
; Remarks:              This routine is called by InitState.
;                       It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*****************************************************************/
;
        public  Step
Step    proc

        mov     eax, [ebp].NextStepCount
        inc     eax
        xor     edx, edx
        mov     ecx, 4
        div     ecx
        mov     [ebp].NextStepCount, edx

        mov     eax, [ebp].SearchLoc
        cmp     [ebp].SearchLocFound, FALSE
        je      DontUseNextStep

        or      edx, edx
        je      StepSetGLOffset
        cmp     edx, 2
        je      StepSetGLOffset

        inc     eax
        cmp     edx, 1
        je      StepDivideBy3
        inc     eax
StepDivideBy3:
        xor     edx, edx
        mov     ecx, 3
        div     ecx
        mov     eax, edx

StepSetGLOffset:
        mov     eax, SearchTbl[eax * 4]
        mov     [ebp].GLOffset, eax
        jmp     StepCalcRx

DontUseNextStep:
        mov     ecx, SearchTbl[eax * 4]
        mov     [ebp].GLOffset, ecx

        inc     eax
        xor     edx, edx
        mov     ecx, 3
        div     ecx
        mov     [ebp].GLOffset, edx

StepCalcRx:

        mov     [ebp].Drift, 0
        call    CalculateRxFreq
        mov     [ebp].Drift, NOM_COUNT_TRACK
        ret

Step    endp
        subttl  -- InitState --
        page
;
;*****************************************************************\
;
; BEGIN_MANUAL_ENTRY( InitState, DPC/API/INITSTA )
;
; Name:         InitState
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*****************************************************************/
;
        public  InitState
InitState       proc

        cmp     [ebp].TrackingMode, TRUE
        jne     InitStateNotTracking

        cmp     DebugMask, 0
        je      InitStateNoMsg
```

```
        mov     eax, offset InitStateTrackMsg
        cmp     eax, LastDebugMessage
        je      InitStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
InitStateNoMsg:

        call    CalculateDriftDelta
        add     eax, NOM_COUNT_REACQ

        mov     edx, [ebp].IOCountNomLowAddr
        out     dx, al
        shr     al, 8
        mov     edx, [ebp].IOCountNomHighAddr
        out     dx, al

        mov     edx, [ebp].IOGateCountHighAddr
        mov     eax, [ebp].ReacqGateCount
        out     dx, al

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, RESET_FEC_ACQ_MASK
        out     dx, al

        mov     edx, [ebp].IOBtrControlAddr
        xor     eax, eax
        out     dx, ax

        mov     edx, [ebp].IOAfcControlAddr
        in      al, dx
        or      al, SWP_ENA_MASK
        out     dx, al

        mov     edx, [ebp].IOBtrControlAddr
        in      al, dx
        or      al, FREQ_PWR_OFFSET OR 6 OR BTR_SENSE_MASK
        out     dx, al

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, RESET_FEC_ACQ_MASK
        out     dx, al

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        cmp     [ebp].ViterbiMode, LOWRATE
        jne     InitStateSetMode
        and     al, NOT MODE_MASK
        jmp     InitStateCheckRate

InitStateSetMode:
        or      al, MODE_MASK
InitStateCheckRate:
        out     dx, al

        mov     edx, [ebp].IOSpareIOControlAddr
        mov     al, 0ah
        cmp     [ebp].ViterbiOnly, 2
        je      InitStateSetRate
        mov     al, 0bh
        cmp     [ebp].ViterbiOnly, 1
        je      InitStateSetRate
        mov     al, 0fh
InitStateSetRate:
        out     dx, al
        mov     [ebp].NextState, ENABLE_BTR
        jmp     InitStateCheckPointing

InitStateNotTracking:

        cmp     DebugMask, 0
        je      InitStateNNoMsg
        mov     eax, offset InitStateNoTrackMsg
        cmp     eax, LastDebugMessage
        je      InitStateNNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
InitStateNNoMsg:

        mov     edx, [ebp].IOBitDetControlAddr
        xor     eax, eax
        out     dx, al

        mov     edx, [ebp].IOSpareIOControlAddr
        mov     al, 0ah
        cmp     [ebp].ViterbiOnly, 2
        je      InitStateNTSetRate
        mov     al, 0bh
        cmp     [ebp].ViterbiOnly, 1
        je      InitStateNTSetRate
        mov     al, 0fh
InitStateNTSetRate:
        out     dx, al

        mov     edx, [ebp].IODaAdOffsetControlAddr
        xor     eax, eax
        out     dx, al

        mov     edx, [ebp].IOAfcControlAddr
        mov     eax, [ebp].ModulationScheme
        or      eax, SWEEP_DIR_SENSE_MASK
        out     dx, al

        mov     edx, [ebp].IOSweepRateAddr
        mov     al, 8ah
        out     dx, al

        mov     edx, [ebp].IOGateCountHighAddr
        mov     eax, [ebp].ReacqGateCount
        out     dx, al

        mov     edx, [ebp].IOCountDeltaAddr
        mov     eax, [ebp].SqfDeltaCount
        out     dx, al

        mov     eax, [ebp].NomCountSearch
        mov     [ebp].TuneCount, eax
        mov     edx, [ebp].IOCountNomLowAddr
        out     dx, al
        shr     eax, 8
        mov     edx, [ebp].IOCountNomHighAddr
        out     dx, al

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
```

```
        or      al, RESET_FEC_ACQ_MASK
        out     dx, al

        mov     edx, [ebp].IOBtrControlAddr
        xor     eax, eax
        out     dx, al

        mov     edx, [ebp].IOAfcControlAddr
        in      al, dx
        or      al, SWP_ENA_MASK
        out     dx, al

        mov     edx, [ebp].IOAgcFirControlAddr
        mov     al, 10 OR AGC_SENSE_MASK
        out     dx, al

        mov     edx, [ebp].IOBtrControlAddr
        in      al, dx
        or      al, FREQ_PWR_OFFSET OR 6 OR BTR_SENSE_MASK
        out     dx, al

        mov     edx, [ebp].IOCrlkThrLowAddr
        mov     al, 60h
        out     dx, al

        mov     edx, [ebp].IOCthAddr
        mov     al, 0e0h
        out     dx, al

        mov     edx, [ebp].IOSynthSerControlAddr
        mov     al, SENA_MASK
        cmp     [ebp].ModulationScheme, BPSK
        jne     InitStateSetSena
        or      al, DEPUNC_BYPASS_MASK
InitStateSetSena:
        out     dx, al

        mov     edx, [ebp].IOCrlkControlAddr
        mov     al, 16 OR CRLK_GAIN_OFFSET OR CRLK_DET_PWR_OFFSET
        out     dx, al

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        cmp     [ebp].ViterbiMode, LOWRATE
        jne     InitStateResetBtr

        or      al, RESET_BTR_ACC_MASK
        and     al, NOT MODE_MASK
        jmp     InitStateClearBtr
InitStateResetBtr:
        or      al, MODE_MASK OR RESET_BTR_ACC_MASK
InitStateClearBtr:
        out     dx, al

        in      al, dx
        and     al, NOT RESET_BTR_ACC_MASK
        out     dx, al

        call    Step

        mov     [ebp].NextState, ACQ_PD

InitStateCheckPointing:

        mov     [ebp].RateCount, 0
        mov     [ebp].PointingFlag, TRUE
        cmp     [ebp].DemodCommand, POINTING_MODE
        je      InitStateExit
        mov     [ebp].PointingFlag, FALSE
InitStateExit:
        mov     [ebp].CurrentState, SYNTH_PRGM
        mov     [ebp].SignalQuality, 0
        mov     [ebp].DemodCommand, BUSY_MODE
        mov     [ebp].DemodStatus, UNLOCKED
        mov     [ebp].FecStatus, UNLOCKED
        ret

InitState       endp
        subttl  -- ProgTuner --
        page
;
;**********************************************************************\
;
; BEGIN_MANUAL_ENTRY( ProgTuner, DPC/API/PROGTUN )
;
; Name:         ProgTuner
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by Tune.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;**********************************************************************/
;
        public  ProgTuner
ProgTuner       proc
;
; EAX = data
; ECX = len
;
        dec     ecx
        mov     edx, 1
        shl     edx, cl
        mov     ecx, edx
        mov     esi, eax                ; ESI = Data
```

```
ProgTunerLoop:
        jecxz   ProgTunerExit

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        test    esi, ecx
        je      ProgTunerClear

        or      al, SDATA_MASK
        and     al, NOT SCLK_MASK
        out     dx, al
        jmp     ProgTunerDelay

ProgTunerClear:
        and     al, NOT (SCLK_MASK OR SDATA_MASK)
        out     dx, al

ProgTunerDelay:
        shr     ecx, 1

        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        in      al, dx

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, SCLK_MASK
        out     dx, al

        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        in      al, dx

        jmp     ProgTunerLoop

ProgTunerExit:
        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT SCLK_MASK
        out     dx, al

        ret

ProgTuner       endp
        subttl  -- Tune --
        page
;
;**********************************************************************\
;
; BEGIN_MANUAL_ENTRY( Tune, DPC/API/TUNE )
;
; Name:         Tune
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by SynthPrgmState.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*********************************************************************/
;
        public  Tune
Tune    proc

        cmp     [ebp].TunerTypeFound, SHARP
        je      TuneSharpPan
        cmp     [ebp].TunerTypeFound, PANASONIC
        je      TuneSharpPan
        cmp     [ebp].TunerTypeFound, SHARP_CUSTOM
        je      TuneSharpCustom
        ret

TuneSharpPan:
        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, SENA_MASK
        out     dx, al

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT (SCLK_MASK OR SDATA_MASK)
        out     dx, al

        cmp     [ebp].TrackingMode, 0
        jne     TuneSetNA

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT SENA_MASK
        out     dx, al

        mov     eax, 2ch
        cmp     [ebp].TunerTypeFound, SHARP
        je      TuneProgTuner
        mov     eax, 0ech
TuneProgTuner:
        mov     ecx, 8
        call    ProgTuner

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, SENA_MASK
        out     dx, al

        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        in      al, dx
```

```
        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT SENA_MASK
        out     dx, al

        mov     eax, 28h OR 2000h
        mov     ecx, 16
        call    ProgTuner

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, SENA_MASK
        out     dx, al

        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        in      al, dx

TuneSetNA:
        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT SENA_MASK
        out     dx, al

        mov     eax, [ebp].ChannelNumber
        add     eax, [ebp].GLDrift
        xor     edx, edx
        mov     ecx, SYNTH_RATIO
        div     ecx
        mov     edi, edx
        or      eax, 3000h

        mov     ecx, 16
        call    ProgTuner

        mov     eax, edi
        mov     ecx, 8
        call    ProgTuner

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, SENA_MASK
        out     dx, al

        ret

TuneSharpCustom:
        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT SENA_MASK
        out     dx, al

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT (SCLK_MASK OR SDATA_MASK)
        out     dx, al

        mov     eax, 50h OR 8001h
        mov     ecx, 16
        call    ProgTuner

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, SENA_MASK
        out     dx, al
        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        in      al, dx

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT SENA_MASK
        out     dx, al

        mov     eax, [ebp].ChannelNumber
        add     eax, [ebp].GLDrift
        xor     edx, edx
        mov     ecx, SYNTH_RATIO
        div     ecx
        mov     edi, edx

        mov     ecx, 11
        call    ProgTuner

        mov     eax, edi
        shl     eax, 1
        mov     ecx, 9
        call    ProgTuner

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, SENA_MASK
        out     dx, al

        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        in      al, dx

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT SENA_MASK
        out     dx, al

        ret

Tune    endp
        subttl  -- SynthPrgmState --
        page
;
;*******************************************************************\
;
; BEGIN_MANUAL_ENTRY( SynthPrgmState, DPC/API/SYNTHPS )
;
; Name:         SynthPrgmState
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
```

```
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*******************************************************************/
;
        public  SynthPrgmState
SynthPrgmState  proc

        cmp     DebugMask, 0
        je      SynthPrgmStateNoMsg
        mov     eax, offset SynthPrgmMsg
        cmp     eax, LastDebugMessage
        je      SynthPrgmStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
SynthPrgmStateNoMsg:
        call    Tune

        mov     [ebp].TrackingMode, 0
        cmp     [ebp].NextState, ACQ_PD
        jne     SynthPrgmClearT2

        mov     [ebp].MaxSqf, 0
        mov     [ebp].SqfAvg, 0
        mov     [ebp].SqfWait, 0
        mov     eax, [ebp].SqfCheckPoints
        mov     [ebp].MaxCount, eax
        mov     [ebp].T2Count, 60
        mov     [ebp].CurrentState, ACQ_PD_DELAY
        ret

SynthPrgmClearT2:
        mov     [ebp].T2Count, 0
        mov     [ebp].CurrentState, ACQ_PD_DELAY
        ret

SynthPrgmState  endp
        subttl  -- AcqPDDelayState --
        page
;
;*******************************************************************\
;
; BEGIN_MANUAL_ENTRY( AcqPDDelayState, DPC/API/ACQPDDS )
;
; Name:         AcqPDDelayState
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*********************************************************************/
;
        public  AcqPDDelayState
AcqPDDelayState proc

        cmp     DebugMask, 0
        je      AcqPDDelayStateNoMsg
        mov     eax, offset AcqPDDelayMsg
        cmp     eax, LastDebugMessage
        je      AcqPDDelayStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
AcqPDDelayStateNoMsg:
        cmp     [ebp].T2Count, 0
        jne     AcqPDDelayExit

        mov     edx, [ebp].IOSweepRateAddr
        mov     al, 87h
        out     dx, al

        mov     edx, [ebp].IOAfcControlAddr
        in      al, dx
        and     al, NOT SQF_PEAK_EN_MASK
        out     dx, al

        mov     eax, [ebp].SqfWait
        mov     [ebp].T2Count, eax

        mov     eax, [ebp].NextState
        mov     [ebp].CurrentState, eax

        mov     edx, [ebp].IOAfcControlAddr
        in      al, dx
        or      al, SQF_PEAK_EN_MASK
        out     dx, al
```

```
AcqPDDelayExit:
        ret

AcqPDDelayState endp
        subttl  -- AcqPDState --
        page
;
;*********************************************************************\
;
; BEGIN_MANUAL_ENTRY( AcqPDState, DPC/API/ACQPDS )
;
; Name:         AcqPDState
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*********************************************************************/
;
        public  AcqPDState
AcqPDState      proc

        cmp     DebugMask, 0
        je      AcqPDStateNoMsg
        mov     eax, offset AcqPDMsg
        cmp     eax, LastDebugMessage
        je      AcqPDStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
AcqPDStateNoMsg:
        cmp     [ebp].T2Count, 0
        jne     AcqPDExit

        xor     eax, eax
        mov     edx, [ebp].IOMaxSqfAddr
        in      al, dx
        add     [ebp].SqfAvg, eax

        cmp     eax, [ebp].MaxSqf
        jbe     AcqPDDecMaxCount

        mov     [ebp].MaxSqf, eax
        mov     eax, [ebp].TuneCount
        mov     [ebp].BestTuneCount, eax

AcqPDDecMaxCount:
        dec     [ebp].MaxCount
        jne     AcqPDMaxCountNotZero

        mov     edx, [ebp].IOSweepRateAddr
        mov     al, 8ah
        out     dx, al

        mov     edx, [ebp].IOCountNomLowAddr
        mov     eax, [ebp].BestTuneCount
        out     dx, al
        shr     eax, 8
        mov     edx, [ebp].IOCountNomHighAddr
        out     dx, al

        mov     edx, [ebp].IOCountDeltaAddr
        mov     eax, [ebp].SqfDeltaCount
        shl     eax, 1
        out     dx, al

        mov     eax, [ebp].SqfAvg
        mov     ecx, [ebp].SqfCheckPoints
        xor     edx, edx
        div     ecx
        add     eax, 2
        mov     [ebp].SqfAvg, eax

        mov     [ebp].T2Count, 40
        mov     [ebp].NextState, ENABLE_BTR
        mov     [ebp].CurrentState, ACQ_PD_DELAY
AcqPDExit:
        ret

AcqPDMaxCountNotZero:
        mov     eax, [ebp].TuneCount
        add     eax, [ebp].SqfCheckStepSize
        mov     [ebp].TuneCount, eax

        mov     edx, [ebp].IOCountNomLowAddr
        out     dx, al

        mov     edx, [ebp].IOCountNomHighAddr
        shr     eax, 8
        out     dx, al

        mov     [ebp].T2Count, 20
        mov     [ebp].CurrentState, ACQ_PD_DELAY
        ret

AcqPDState      endp
        subttl  -- EnableBTRState --
        page
;
;*****************************************************************************\
;
; BEGIN_MANUAL_ENTRY( EnableBTRState, DPC/API/ENBTRST )
```

```
;
; Name:          EnableBTRState
;
; Description:   Acquisition State Routine.
;
; On Entry:      EAX     N/A
;                EBX     Frame Data Space
;                ECX     N/A
;                EDX     N/A
;                EBP     Adapter Data Space
;                ESI     N/A
;                EDI     N/A
;
;                Note:   Interrupts are in any state.
;
; On Return:     EAX     Destroyed
;                EBX     Preserved
;                ECX     Destroyed
;                EDX     Destroyed
;                EBP     Preserved
;                ESI     Preserved
;                EDI     Preserved
;
;                Flags:
;
;                Note:   Interrupts preserved.
;
; Remarks:       This routine is called by DriverCallBack.
;                It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*****************************************************************************/
;
        public  EnableBTRState
EnableBTRState  proc

        cmp     DebugMask, 0
        je      EnableBTRStateNoMsg
        mov     eax, offset EnableBTRMsg
        cmp     eax, LastDebugMessage
        je      EnableBTRStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
EnableBTRStateNoMsg:
        mov     edx, [ebp].IOBtrControlAddr
        in      al, dx
        or      al, BTR_ERR_ENA_MASK
        out     dx, al

        mov     [ebp].CurrentState, START_SEARCH_FOR_FEC
        ret

EnableBTRState  endp
        subttl  -- StartSearchForFECState --
        page
;
;*****************************************************************************\
;
; BEGIN_MANUAL_ENTRY( StartSearchForFECState, DPC/API/SRCHFEC )
;
```

```
; Name:         StartSearchForFECState
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*******************************************************************/
;
        public  StartSearchForFECState
StartSearchForFECState  proc

        cmp     DebugMask, 0
        je      StartSearchForFECStateNoMsg
        mov     eax, offset StartSearchForFECMsg
        cmp     eax, LastDebugMessage
        je      StartSearchForFECStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
StartSearchForFECStateNoMsg:
        cmp     [ebp].PointingFlag, 0
        je      SearchFECNotPointing

        mov     [ebp].CurrentState, POINTING_ACQ
        mov     eax, [ebp].SqfAvg
        add     eax, 2
        mov     [ebp].MaxSqf, eax
        jmp     SearchFECSetMax

SearchFECNotPointing:

        mov     [ebp].CurrentState, CHECK_FOR_FEC_LOCK
        mov     eax, [ebp].SqfAvg
        add     eax, 6
        mov     [ebp].MaxSqf, eax

SearchFECSetMax:
```

```
        mov     edx, [ebp].IOCthAddr
        out     dx, al

        mov     edx, [ebp].IOAfcControlAddr
        in      al, dx
        and     al, NOT SWP_ENA_MASK
        out     dx, al

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, RESET_FEC_ACQ_MASK
        out     dx, al

        mov     [ebp].T1Count, 300

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT RESET_FEC_ACQ_MASK
        out     dx, al

        ret

StartSearchForFECState  endp
        subttl  -- CheckforFECLockState --
        page
;
;*******************************************************************\
;
; BEGIN_MANUAL_ENTRY( CheckforFECLockState, DPC/API/CHKFEC )
;
; Name:         CheckforFECLockState
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*******************************************************************/
;
        public  CheckforFECLockState
```

```
CheckforFECLockState    proc

        cmp     DebugMask, 0
        je      CheckForFECLockStateNoMsg
        mov     eax, offset CheckforFECLockStateMsg
        cmp     eax, LastDebugMessage
        je      CheckForFECLockStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
CheckForFECLockStateNoMsg:

        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        and     al, FEC_LOCK_MASK
        je      CheckFECNotLocked

        mov     [ebp].DemodStatus, LOCKED
        mov     [ebp].FecStatus, LOCKED

        mov     edx, [ebp].IOGateCountHighAddr
        xor     eax, eax
        out     dx, al

        mov     edx, [ebp].IOCountDeltaAddr
        mov     eax, [ebp].ReacqDeltaCount
        out     dx, al

        mov     edx, [ebp].IOBtrControlAddr
        in      al, dx
        and     al, NOT (FREQ_PWR_MASK OR PHASE_PWR_MASK)
        out     dx, al

        in      al, dx
        or      al, 5
        out     dx, al

        mov     [ebp].NextStepCount, 1
        mov     [ebp].T1Count, 500
        mov     [ebp].T2Count, 100
        mov     [ebp].CurrentState, TRACKING
        ret

CheckFECNotLocked:

        cmp     [ebp].T1Count, 0
        jne     CheckFECHaveT1Count

        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        test    al, CRL_LOCK_MASK
        jne     CheckFECSetOtherMode

        mov     [ebp].CurrentState, INIT
        ret

CheckFECSetOtherMode:
        mov     [ebp].CurrentState, SET_OTHER_MODE
CheckFECExit:
        ret

CheckFECHaveT1Count:
        mov     edx, [ebp].IOStatusAddr
        in      al, dx
```

```
        test    al, CRL_LOCK_MASK
        jne     CheckFECExit
        mov     eax, [ebp].MaxSqf
        cmp     eax, [ebp].SqfAvg
        jbe     CheckFECExit
        sub     eax, 2
        mov     [ebp].MaxSqf, eax
        mov     edx, [ebp].IOCthAddr
        out     dx, al
        ret

CheckforFECLockState    endp
        subttl  -- SetOtherModeState --
        page
;
;*****************************************************************************\
;
; BEGIN_MANUAL_ENTRY( SetOtherModeState, DPC/API/SETOTHER )
;
; Name:         SetOtherModeState
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*****************************************************************************/
;
        public  SetOtherModeState
SetOtherModeState       proc

        cmp     DebugMask, 0
        je      SetOtherModeStateNoMsg
        mov     eax, offset SetOtherModeStateMsg
        cmp     eax, LastDebugMessage
        je      SetOtherModeStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
SetOtherModeStateNoMsg:

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        cmp     [ebp].ViterbiMode, LOWRATE
        jne     SetOtherModeToLow

        or      al, MODE_MASK
        out     dx, al

        mov     [ebp].ViterbiMode, HIGHRATE
        jmp     SetOtherModeIncRate

SetOtherModeToLow:
        and     al, NOT MODE_MASK
        out     dx, al

        mov     [ebp].ViterbiMode, LOWRATE

SetOtherModeIncRate:
        inc     [ebp].RateCount

        cmp     [ebp].RateCount, 1
        jg      SetOtherModeExit

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, RESET_FEC_ACQ_MASK
        out     dx, al

        mov     [ebp].T1Count, 180

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT RESET_FEC_ACQ_MASK
        out     dx, al

        mov     [ebp].CurrentState, CHECK_FOR_FEC_LOCK
        ret

SetOtherModeExit:
        mov     [ebp].CurrentState, INIT
        ret

SetOtherModeState       endp
        subttl  -- ReadWord --
        page
;
;*******************************************************************\
;
; BEGIN_MANUAL_ENTRY( ReadWord, DPC/API/READWORD )
;
; Name:         ReadWord
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
```

```
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by TrackingState,
;               PointingAcquisitionState and PointingTrackingState
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*******************************************************************/
;
        public  ReadWord
ReadWord        proc

        push    ebx

        xor     ebx, ebx
        mov     edx, edi
        in      al, dx
        mov     bh, al

        mov     ecx, 4
ReadWordLoop:
        mov     edx, esi
        in      al, dx
        mov     bl, al

        mov     edx, edi
        in      al, dx
        cmp     al, bh
        je      ReadWordExit

        mov     bh, al
        dec     ecx
        jne     ReadWordLoop

ReadWordExit:
        mov     eax, ebx

        pop     ebx
        ret

ReadWord        endp
        subttl  -- TrackingState --
        page
;
;*******************************************************************\
;
; BEGIN_MANUAL_ENTRY( TrackingState, DPC/API/TRCKST )
;
; Name:         TrackingState
;
; Description:  Acquisition State Routine.
;
```

```
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*****************************************************************/.
;
        public  TrackingState
TrackingState   proc

        cmp     DebugMask, 0
        je      TrackingStateNoMsg
        mov     eax, offset TrackingStateMsg
        cmp     eax, LastDebugMessage
        je      TrackingStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
TrackingStateNoMsg:

        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        test    al, FEC_LOCK_MASK
        jne     TrackingStateReadQuality

        in      al, dx
        test    al, CRL_LOCK_MASK
        je      TrackingStateZero
        cmp     [ebp].T2Count, 0
        jne     TrackingStateT1

TrackingStateZero:
        mov     [ebp].TrackingMode, 0
        mov     [ebp].CurrentState, INIT
        ret

TrackingStateT1:
        mov     [ebp].T1Count, 1000
        ret
;
```

```
TrackingStateReadQuality:
        xor     eax, eax
        mov     edx, [ebp].IORelSqfAddr
        in      al, dx
        mov     [ebp].SignalQuality, eax

        cmp     DebugMask, 0
        je      SignalStrengthNoMsg
        cmp     LastSignalStrength, 0
        jne     SignalStrengthNoMsg

        mov     LastSignalStrength, 1
        cmp     eax, 200
        jb      SignalStrengthNone
        sub     eax, 200
        shl     eax, 1
        add     eax, 60
        jmp     SignalStrengthPrint
SignalStrengthNone:
        xor     eax, eax
SignalStrengthPrint:
        push    eax
        push    offset SignalStrengthMsg
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (3 * 4)]

SignalStrengthNoMsg:

        cmp     [ebp].T1Count, 0
        jne     TrackingStateExit

        mov     [ebp].T1Count, 1000
        mov     [ebp].TrackingMode, 1

        mov     edi, [ebp].IOTuningHighAddr
        mov     esi, [ebp].IOTuningLowAddr
        call    ReadWord
        mov     [ebp].Drift, eax

        mov     ecx, 2
        cmp     eax, NOM_COUNT_TRACK + OFFSET_THRESHOLD
        ja      TrackingStateLocFound
        mov     ecx, 0
        cmp     eax, NOM_COUNT_TRACK - OFFSET_THRESHOLD
        jb      TrackingStateLocFound
        mov     ecx, 1
TrackingStateLocFound:
        mov     [ebp].SearchLoc, ecx
        mov     [ebp].SearchLocFound, TRUE
TrackingStateExit:
        ret

TrackingState   endp
        subttl  -- PointingAcquisitionState --
        page
;
;*****************************************************************\
;
; BEGIN_MANUAL_ENTRY( PointingAcquisitionState, DPC/API/PTACQST )
;
; Name:         PointingAcquisitionState
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;********************************************************************/
;
        public  PointingAcquisitionState
PointingAcquisitionState        proc

        cmp     DebugMask, 0
        je      PointingAcquisitionStateNoMsg
        mov     eax, offset PointingAcqStateMsg
        cmp     eax, LastDebugMessage
        je      PointingAcquisitionStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
PointingAcquisitionStateNoMsg:

        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        test    al, SWEEPING_MASK
        je      PointingNotSweeping

        mov     esi, [ebp].IOTuningLowAddr
        mov     edi, [ebp].IOTuningHighAddr
        call    ReadWord
        shl     eax, 4
        mov     [ebp].Drift, eax

        mov     [ebp].DemodStatus, LOCKED

        xor     eax, eax
        mov     edx, [ebp].IOGateCountHighAddr
        out     dx, al

        mov     edx, [ebp].IOBtrControlAddr
        in      al, dx
        and     al, NOT (FREQ_PWR_MASK OR PHASE_PWR_MASK)
        out     dx, al
```

```
        in      al, dx
        or      al, 5
        out     dx, al

        mov     [ebp].NextStepCount, 1
        mov     [ebp].T1Count, 1000
        mov     [ebp].SearchLocFound, FALSE
        mov     [ebp].CurrentState, POINTING_TRACKING
        ret

PointingNotSweeping:
        mov     eax, [ebp].MaxSqf
        sub     eax, 2
        mov     [ebp].MaxSqf, eax
        mov     edx, [ebp].IOCthAddr
        out     dx, al
        cmp     [ebp].T1Count, 0
        jne     PointingAcqExit

        mov     [ebp].CurrentState, INIT

PointingAcqExit:
        ret

PointingAcquisitionState        endp
        subttl  -- PointingTrackingState --
        page
;
;********************************************************************\
;
; BEGIN_MANUAL_ENTRY( PointingTrackingState, DPC/API/PTTRKST )
;
; Name:         PointingTrackingState
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
```

```
;***********************************************************************/
;
        public  PointingTrackingState
PointingTrackingState   proc

        cmp     DebugMask, 0
        je      PointingTrackingStateNoMsg
        mov     eax, offset PointingTrackStateMsg
        cmp     eax, LastDebugMessage
        je      PointingTrackingStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
PointingTrackingStateNoMsg:
        xor     eax, eax
        mov     edx, [ebp].IORelSqfAddr
        in      al, dx
        mov     [ebp].SignalQuality, eax

        cmp     [ebp].T1Count, 0
        je      PointingTrackingExit

        mov     [ebp].T1Count, 1000

        mov     esi, [ebp].IOTuningLowAddr
        mov     edi, [ebp].IOTuningHighAddr
        call    ReadWord

        mov     ecx, [ebp].Drift
        add     ecx, OFFSET_THRESHOLD
        cmp     eax, ecx
        ja      PointingTrackingInit

        mov     ecx, [ebp].Drift
        sub     ecx, OFFSET_THRESHOLD
        cmp     eax, ecx
        jae     PointingTrackingExit

PointingTrackingInit:
        mov     [ebp].CurrentState, INIT

PointingTrackingExit:
        ret

PointingTrackingState   endp
        subttl  -- HaltState --
        page
;
;***********************************************************************\
;
; BEGIN_MANUAL_ENTRY( HaltState, DPC/API/HALTST )
;
; Name:         HaltState
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
```

```
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;***********************************************************************/
;
        public  HaltState
HaltState       proc

        cmp     DebugMask, 0
        je      HaltStateNoMsg
        mov     eax, offset HaltStateMsg
        cmp     eax, LastDebugMessage
        je      HaltStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
HaltStateNoMsg:
        ret

HaltState       endp
        subttl  -- InitDemod --
        page
;
;***********************************************************************\
;
; BEGIN_MANUAL_ENTRY( InitDemod, DPC/API/INITDMOD )
;
; Name:         InitDemod
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*******************************************************************/
;
        public  InitDemod
InitDemod       proc

        mov     [ebp].CurrentState, HALT
        mov     [ebp].RxFreq, 0
        mov     [ebp].ViterbiMode, 0
        mov     [ebp].DemodCommand, HALT_MODE
        mov     [ebp].SearchLoc, 1
        mov     [ebp].Drift, 0
        mov     [ebp].GLOffset, 0
        mov     [ebp].TrackingMode, FALSE


        ;value = read_bits (STATUS_ADDR, TUNER_TYPE_MASK);
        xor     eax, eax
        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        and     al, TUNER_TYPE_MASK
        mov     cl, al

        ;value |= read_bits (UNIT_ID_ADDR, TUNER_TYPE_2_MASK);
        mov     edx, [ebp].IOUnitIDAddr
        in      al, dx
        and     al, TUNER_TYPE_2_MASK
        or      al, cl

        cmp     al, SHARP
        jne     InitDemodPanasonic

        cmp     DebugMask, 0
        je      FillInTunerVars
        push    eax
        push    offset SharpTunerMsg
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
        pop     eax
        jmp     FillInTunerVars

InitDemodPanasonic:
        cmp     al, PANASONIC
        jne     InitDemodSharpCustom

        cmp     DebugMask, 0
        je      FillInTunerVars
        push    eax
        push    offset PanasonicTunerMsg
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
```

```
        pop     eax
        jmp     FillInTunerVars

InitDemodSharpCustom:
        cmp     al, SHARP_CUSTOM
        jne     InitDemodExit

        cmp     DebugMask, 0
        je      FillInTunerVars
        push    eax
        push    offset SharpCustomTunerMsg
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
        pop     eax

FillInTunerVars:
        mov     [ebp].TunerTypeFound, eax
        mov     [ebp].ReacqGateCount, 0f0h
        mov     [ebp].ReacqDeltaCount, 2
        mov     [ebp].NomCountSearch, NOM_COUNT_REACQ - 75
        mov     [ebp].SqfCheckPoints, 11
        mov     [ebp].SqfCheckStepSize, 15
        mov     [ebp].SqfDeltaCount, 8
InitDemodExit:
        ret

InitDemod       endp
        subttl  -- ApplyDelay --
        page
;
;*******************************************************************\
;
; BEGIN_MANUAL_ENTRY( ApplyDelay, DPC/API/APPLYDEL )
;
; Name:         ApplyDelay
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
```

```
; END_MANUAL_ENTRY
;
;*****************************************************************/
;
        public  ApplyDelay
ApplyDelay      proc

;
; Apply delay to T1 Counter
;
        cmp     [ebp].T1Count, 0
        je      ApplyDelayT2
        cmp     [ebp].T1Count, eax
        jb      ApplyDelayClearT1
        sub     [ebp].T1Count, eax
        jmp     ApplyDelayT2
ApplyDelayClearT1:
        mov     [ebp].T1Count, 0
;
; Apply delay to T2 Counter
;
ApplyDelayT2:
        cmp     [ebp].T2Count, 0
        je      ApplyDelayExit
        cmp     [ebp].T2Count, eax
        jb      ApplyDelayClearT2
        sub     [ebp].T2Count, eax
        jmp     ApplyDelayExit
ApplyDelayClearT2:
        mov     [ebp].T2Count, 0

ApplyDelayExit:
        ret

ApplyDelay      endp
        subttl  -- CalculateRxFreq --
        page
;
;*****************************************************************\
;
; BEGIN_MANUAL_ENTRY( CalculateRxFreq, DPC/API/CALCRXFQ )
;
; Name:         CalculateRxFreq
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
```

```
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*****************************************************************/
;
        public  CalculateRxFreq
CalculateRxFreq proc

        ;  USHORT_T freq, total, new_total, results;
        sub     esp, 2 * 4

        ; total = [esp + 0]
        ; results = [esp + 4]
        ; freq = esi
        ; new_total = edi

        ;   freq = S.Rx_Freq - FREQ_BASE;
        ;   freq += S.GL_offset;
        mov     esi, [ebp].RxFreq
        sub     esi, FREQ_BASE
        add     esi, [ebp].GLOffset     ; ESI = freq

        ;   total = (freq * 2) / 729;
        mov     eax, esi                ; EAX = freq
        shl     eax, 1                  ; EAX = freq * 2
        xor     edx, edx
        mov     ecx, 729
        div     ecx                     ; EAX = EAX / 729
        mov     [esp + 0], eax          ; total = EAX

        ;   results = (total * 729) / 2;
        mov     ecx, 729
        mul     ecx                     ; EAX = total * 729
        shr     eax, 1                  ; EAX = EAX / 2
        mov     [esp + 4], eax          ; results = eax

        ;   freq = freq - results;
        sub     esi, eax

        ;   new_total = total * 10;
        mov     eax, [esp + 0]
        mov     ecx, 10
        mul     ecx                     ; EAX = total * 10
        mov     edi, eax                ; new_tottal = EAX

        ;   total = (freq * 20) / 729;
        mov     eax, esi                ; EAX = freq
        mov     ecx, 20
        mul     ecx                     ; EAX = freq * 20
        xor     edx, edx
        mov     ecx, 729
        div     ecx                     ; EAX = EAX / 729
        mov     [esp + 0], eax          ; total = EAX

        ;   results = (total * 729) / 20;
        mov     ecx, 729
        mul     ecx                     ; EAX = total * 729
        xor     edx, edx
        mov     ecx, 20
        div     ecx                     ; EAX = EAX / 20
        mov     [esp + 4], eax          ; results = EAX

        ;   freq = freq - results;
        sub     esi, eax

        ;   new_total += total;
        add     edi, [esp + 0]

        ;   new_total *= 10;
        mov     eax, edi
        mov     ecx, 10
        mul     ecx                     ; EAX = new_total * 10
        mov     edi, eax                ; EDI = EAX

        ;   total = (freq * 200) / 729;
        mov     eax, esi
        mov     ecx, 200
        mul     ecx                     ; EAX = freq * 200
        xor     edx, edx
        mov     ecx, 729
        div     ecx                     ; EAX = EAX / 729
        mov     [esp + 0], eax          ; total = EAX

        ;   results = (total * 729) / 200;
        mov     ecx, 729
        mul     ecx                     ; EAX = total * 729
        xor     edx, edx
        mov     ecx, 200                ; EAX = EAX / 200
        div     ecx
        mov     [esp + 4], eax          ; results = EAX

        ;   freq = freq - results;
        sub     esi, eax

        ;   new_total += total;
        add     edi, [esp + 0]

        ;   if (freq >= 2) new_total++;
        cmp     esi, 2
        jb      CalcGetChannelNumber
        inc     edi                     ; new_total++
CalcGetChannelNumber:

        ;   S.Channel_Number = SYNTH_FIRST_CHANNEL + new_total;
        add     edi, SYNTH_FIRST_CHANNEL
        mov     [ebp].ChannelNumber, edi

        cmp     DebugMask, 0
        je      NoChannelMsg
        push    edi
        push    offset ChannelNumberMsg
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (3 * 4)]
NoChannelMsg:

        add     esp, 2 * 4
        ret

CalculateRxFreq endp
        subttl  -- DriverCallBack --
        page
;
;*****************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverCallBack, DPC/API/CALLBACK )
```

```
;
; Name:           DriverCallBack
;
; Description:    This routine will be executed once every second. It will
;                 detect if the hardware does not ack a transmission. If the
;                 hardware didn't ack then it will be reset, the transmission
;                 of that packet will be aborted and the next packet in the
;                 queue will be sent if there is one.
;
; On Entry:       EAX     N/A
;                 EBX     @ Frame Data Space
;                 ECX     N/A
;                 EDX     N/A
;                 EBP     @ Adapter Data Space
;                 ESI     N/A
;                 EDI     N/A
;
;                 Note:   Interrupts are disabled.
;
; On Return:      EAX     Destroyed
;                 EBX     Preserved
;                 ECX     Destroyed
;                 EDX     Destroyed
;                 EBP     Preserved
;                 ESI     Destroyed
;                 EDI     Destroyed
;
;                 Flags:
;
;                 Note:   Interrupts disabled.
;
; Remarks:        This routine is called by the MSM.
;                 After this call returns, the MSM will schedule another
;                 call back.
;                 It is called at interrupt time.
;
; See Also:       MSM\MSMCallBackProcedure
;
; END_MANUAL_ENTRY
;
;*****************************************************************/
;
        public  DriverCallBack
        align   16
DriverCallBack  proc

        cmp     [ebp].TunerTypeFound, INVALID_TUNER
        jne     DemodInitialized
        call    InitDemod
;
; Check Rx Frequency
;
DemodInitialized:
;;      cmp     [ebp].RxFreq, 1330 * 10
;;      je      CallBackCheckState
;;      mov     [ebp].RxFreq, 1330 * 10

        cmp     [ebp].RxFreq, 0
        jne     CallBackCheckState
        mov     eax, GlobalRxFreq
;
; RxFreq = GlobalRxFreq * 10
;
        mov     ecx, 10
        mul     ecx
        mov     [ebp].RxFreq, eax
        cmp     [ebp].TrackingMode, FALSE
        je      CheckRxFreqSetMode
        call    CalculateRxFreq
CheckRxFreqSetMode:
        mov     [ebp].DemodCommand, ACQUIRE_MODE
;
; Possibly set the CurrentState depending on DemodCommand
;
CallBackCheckState:
        cmp     [ebp].DemodCommand, ACQUIRE_MODE
        je      CallBackSetInit
        cmp     [ebp].DemodCommand, POINTING_MODE
        jne     CallBackCheckHalt
CallBackSetInit:
        mov     [ebp].CurrentState, INIT
        call    CallBackApplyDelay
        jmp     CallBackWatchDog

CallBackCheckHalt:
        cmp     [ebp].DemodCommand, HALT_MODE
        jne     CallBackApplyDelay
        mov     [ebp].CurrentState, HALT
;
; Apply delay
;
CallBackApplyDelay:
        mov     eax, 15
        call    ApplyDelay

        mov     eax, [ebp].CurrentState
        mov     esi, StateTbl[eax * 4]
        call    esi

CallBackWatchDog:

        ret

        mov     eax, [ebp].BufferCount
        cmp     eax, [ebp].WatchBufferCount
        mov     [ebp].WatchBufferCount, eax
        jne     CallBackExit

        call    RefreshMipsStats

        mov     eax, [ebp].MipsZeroAddrFrames           ; LocalMipsStats.zeroAdd
rFrames
        cmp     eax, [ebp].WatchOldRejected
        mov     [ebp].WatchOldRejected, eax
        je      CallBackExit

        call    DriverISR

        mov     edx, [ebp].PicAddress
        in      al, dx
        SLOW
        or      eax, [ebp].PicMask
        out     dx, al
        SLOW
        in      al, dx
        SLOW
        and     eax, [ebp].PicUnMask
        out     dx, al

CallBackExit:
        ret
```

```
DriverCallBack  endp
        public DriverSend
        subttl  -- DriverSend --
        page
;
;*****************************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverSend, DPC/API/SEND )
;
; Name:         DriverSend
;
; Description:  This routine will transfer the packet described in the
;               TCB to the NIC and initiate the send. TxStartTime and
;               RetryCounter must be set to enable the deadman timer.
;
; On Entry:     EAX     N/A
;               EBX     @ Frame Data Space
;               ECX     Padded Packet Length
;               EDX     N/A
;               EBP     @ Adapter Data Space
;               ESI     @ TCB
;               EDI     N/A
;
;               Note:   Interrupts are disabled.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Destroyed
;               EDI     Destroyed
;
;               Flags:
;
;               Note:   Interrupts disabled.
;
; Remarks:      This routine is called by the MSM media module.
;               It is called at process or interrupt time.
;
; See Also:     ETHERTSM\EtherTSMDriverSend
;               ETHERTSM\MediaSendRaw8023
;               ETHERTSM\MediaSendEthernetII
;               ETHERTSM\MediaSend8022Over8023
;               ETHERTSM\MediaSend8022Snap
;
; END_MANUAL_ENTRY
;
;*****************************************************************************/
;
        align   16
DriverSend      proc

        lea     edi, [esi].TCBMediaHeader
        cmp     word ptr [edi+12], 0608h        ; ARP(0x08 0x06)?
        je      DriverSendArp                   ; Jump if it is

        cmp     [ebp].AgentSendRoutine, 0       ; Can we send it yet?
        je      DriverSendExit                                          ; Give i
t back if we can't

        push    ecx
; Padded size
        push    esi
; Address of TCB
        call    [ebp].AgentSendRoutine  ; Give it to Slip Handler
        pop     esi
        pop     ecx

        ret

DriverSendExit:
        inc     [ebp].MSMTxFreeCount            ; Add a send resource
        jmp     EtherTSMFastSendComplete        ; Otherwise service events.

DriverSendArp:
;
; We're going to assume that the entire request is in the first
; fragment. Verify it first.
;
        mov     edi, [esi].TCBFragStrucPtr
        cmp     dword ptr [edi+0], 1            ; One fragment?
        jne     DriverSendExit          ; Jump if not
        cmp     dword ptr [edi+8], 28   ; Entire ARP request(frag length)?
        jb      DriverSendExit          ; Jump if not
;
; Make sure sender and target ip addr are different
;
        mov     edi, [edi+4]                    ; EDI -> ARP request

        mov     eax, [edi+28-14]                ; EAX = senders IP
        cmp     eax, [edi+38-14]                ; Same as target IP?
        je      DriverSendExit          ; Jump out if it is

        push    esi                             ; Save send ECB
        push    edi                             ; Save ARP offset

        mov     esi, 1514                       ; Max ECB size.
        call    MSMAllocateRCB          ; Get an ECB
        pop     edi                             ; EDI -> ARP request again
        or      eax, eax
        jne     DriverSendReturnARP             ; Jump if no ECB.
;
; ESI -> reply ECB
; EDI -> request data
;
        lea     eax, [esi+RPacketEnvelope]      ; EAX -> beginning
        mov     [esi].RPacketOffset, eax        ; Store into ECB
        mov     [esi].RPacketSize, 60   ; ARP reply size
        mov     [esi].RPacketLength, 60 ;  (ethernet min size)

        push    esi                             ; Save reply ECB
        mov     esi, eax                        ; ESI -> reply data
;
; ESI -> reply data
; EDI -> request data
;
;
; Set reply->dest_addr to our node address
;
        mov     eax, dword ptr [ebx].MLIDNodeAddress+0
        mov     dword ptr [esi+0], eax
        mov     ax, word ptr [ebx].MLIDNodeAddress+4
        mov     word ptr [esi+4], ax
;
; Set reply->source_addr to (0x06 0x06 0x06 0x06 0x06 0x06)
;
        mov     word ptr [esi+6], 0606h
        mov     dword ptr [esi+8], 06060606h
;
; Set reply->type to (0x08 0x06)
;
```

```
        mov     word ptr [esi+12], 0608h
;
; Set reply hardware type(0x00 0x01), protocol type(0x08 0x00)
;
        mov     dword ptr [esi+14], 00080100h
;
; Set reply hardware size(0x06), protocol size(0x04)
;    and operation(0x00 0x02 for ARP reply)
;
        mov     dword ptr [esi+18], 02000406h
;
; Set reply senders ethernet addr to (0x06 0x06 0x06 0x06 0x06 0x06).
;
        mov     dword ptr [esi+22], 06060606h
        mov     word ptr [esi+26], 0606h
;
; Set reply senders ip addr to the request target ip addr
;
        mov     eax, [edi+38-14]                ; request->target_ip
        mov     [esi+28], eax
;
; Set reply target ethernet addr to our node addr
;
        mov     eax, dword ptr [ebx].MLIDNodeAddress+0
        mov     dword ptr [esi+32], eax
        mov     ax, word ptr [ebx].MLIDNodeAddress+4
        mov     word ptr [esi+36], ax
;
; Set reply target ip addr to request senders ip addr
;
        mov     eax, dword ptr [edi+28-14]      ; request->senders_ip
        mov     dword ptr [esi+38], eax

        pop     esi                             ; ESI -> reply ECB
        mov     edi, 1514                       ; Max Packet size
        xor     eax, eax                        ; Good packet
        mov     ecx, [esi].RPacketSize
        push    ebp
        call    EtherTSMFastProcessGetRCB
        pop     ebp
        jne     DriverSendReturnARP             ; Jump if no new ECB
        MSMReturnRCB                            ; Return newly allocated ECB

DriverSendReturnARP:
        pop     esi                             ; Restore send ECB
        inc     [ebp].MSMTxFreeCount            ; Add a send resource
        jmp     EtherTSMFastSendComplete        ; Otherwise service events.

DriverSend      endp

        extrn   DoEndOfInterrupt: near
        extrn   SetHardwareInterrupt: near
        extrn   ClearHardwareInterrupt: near
TestDriverISR   proc

        mov     ebp, OurAdapterDataSpace
        mov     ebx, [ebp].MSMDefaultVirtualBoard
        movzx   ecx, [ebx].MLIDInterrupt
        call    DoEndOfInterrupt

        inc     [ebp].GotInterrupt

        xor     eax, eax
        ret

TestDriverISR   endp
```

```
        subttl  -- DriverISR --
        page
;
;*****************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverISR, DPC/API/ISR )
;
; Name:         DriverISR
;
; Description:  This routine handles packet reception.
;
; On Entry:     EAX     N/A
;               EBX     N/A
;               ECX     N/A
;               EDX     N/A
;               EBP     @ Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are disabled.
;
; On Return:    EAX     Destroyed
;               EBX     Destroyed
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Destroyed
;               ESI     Destroyed
;               EDI     Destroyed
;
;               Flags:
;
;               Note:   Interrupts disabled.
;
; Remarks:      This routine is called by the MSM.
;               It is called at interrupt time.
;
; See Also:     MSM\MSMInterruptProcedure
;
; END_MANUAL_ENTRY
;
;*****************************************************************/
;
        align   16
        public  DriverISR
DriverISR       proc
;
; /* Set the adapters ram ptr to the next rbd to receive from */
; outport(bicd_base_addr + MSG_RAM_PTR, rbd_base_addr + 2*curr_adap_rbd);
;
        DebugMessage    DEBUG_ISR_ALL, ISREnterMsg
        mov     edx, [ebp].IOMsgRamPtr                  ; MsgRamPtr I/O port
        mov     eax, [ebp].CurrentAdapterRBD            ; Next Adapter RBD
        shl     eax, 1                                  ;  * 2
        add     eax, RBD_BASE_ADDR                      ; add Base(0a000h)
        out     dx, ax                                  ; Set Adapter Ram Ptr
;
; /* Keep processing packets until no more are left */
; /*
; * NOTE: We are assuming that anyone looping back to DriverISRLoop
; * has set the MSR_RAM_PTR to the next RBD to examine.
; */
; while ((status = inport (bicd_base_addr + MSG_RAM)) & EMPTY)
;
DriverISRLoop:
        xor     eax, eax                                ; Clear upper status
```

```
        mov     edx, [ebp].IOMsgRam                     ; MsgRam I/O port
        in      ax, dx                                  ; Get status
        mov     [ebp].IntStatus, eax                    ; Save off RBD status

        test    eax, EMPTY                              ; This on ready?
        je      DriverISRExit                           ; Jump if its not ready

        inc     [ebp].BufferCount                       ; Used for watchdog
        DebugMessage1   DEBUG_ISR, DebugRBDReceived, [ebp].CurrentAdapterRBD

;
;    /* Jump if this is an error packet */
;    if (status & 0x8F)
;
        test    [ebp].IntStatus, STATUS_ERROR           ; Any error bits set?
        jne     DriverISRBadPacket                      ; Jump if not
;
;    /* Heres a good packet. See if we have a buffer for it. */
;    if (!global_pool[curr_rbd].buf_ptr)
;
        mov     esi, [ebp].CurrentECB                   ; Is this more of
        or      esi, esi                                ;  the last packet?
        jne     DriverISRAddToECB                       ; Jump if it is.
if TIMESTAMP
;       mov     al, 'r'
;       push    eax
;       call    DPCTimestamp
;       lea     esp, [esp + 4]
endif

        mov     esi,1514                                ; Max ECB size.
        call    MSMAllocateRCB                          ; Get an ECB
        or      eax, eax
        jne     DriverISRNoECB                          ; Jump if no ECB.

;
; !!!! Satelite header is 12 bytes, EII is 14 bytes.
; !!!! Add 2 to offset to prevent double copy of turbo internet packets.
;
        lea     edi, [esi+RPacketEnvelope+2]    ; EDI -> beginning
        mov     [esi].RPacketOffset, edi                ; Store into ECB
        mov     [esi].RPacketSize, 0                    ; Clear size
        mov     [ebp].CurrentECB, esi                   ; Store if split packet
        jmp     short DriverISRReadSize

DriverISRAddToECB:
        mov     edi, [esi].RPacketOffset
        add     edi, [esi].RPacketSize

DriverISRReadSize:
;
;    /* ESI(curr_rbd) will be used a lot. Let's try to keep it intact. */
;    /* Retrieve the length of the packet */
;    length = inport(bicd_base_addr + MSG_RAM);
;
        xor     eax, eax                                ; Clear upper word
        mov     edx, [ebp].IOMsgRam                     ; Msg Ram I/O port
        in      ax, dx                                  ; Get size of packet

        add     [esi].RPacketSize, eax                  ; Add to ECB size

        DebugMessage1   DEBUG_ISR, DebugRBDSize, eax

;
;    word_length = (length & 3) ? (length / 4) + 1 : (length/4);
;
```

```
        mov     ecx, eax                                ; ECX = packet length

        shr     ecx, 2                                  ; ECX = ECX / 4
        test    eax, 3                                  ; Any left over?
        jz      DriverISRCopyPacket                     ; Jump if not
        inc     ecx                                     ; read another dword

;;      shr     ecx, 1
;;      test    eax, 1
;;      jz      DriverISRCopyPacket
;;      inc     ecx
DriverISRCopyPacket:
;
;   outport(bicd_base_addr + AUTO_INC, curr_adap_rbd * stats.config.buf_size /
2);
;
        push    ecx
        mov     eax, [ebp].CurrentAdapterRBD            ; EAX = current adapter
RBD
;       mov     ecx, RBD_BUFFER_SIZE                    ; ECX = buffer size
        mov     ecx, RBD_BUFFER_SIZE / 2                ; ECX = buffer size
        mul     ecx                                     ; EAX = EAX * ECX
;       shr     eax, 1                                  ; EAX = EAX / 2
        mov     edx, [ebp].IOAutoInc                    ; Auto Inc I/O port
        out     dx, ax                                  ; Set adapter buffer ptr
        pop     ecx
;
;   Read from RxData port really fast(this was inline assembly)
;
        cld                                             ; Forward march!
        mov     edx, [ebp].IORxData                     ; Rx Data I/O port
        rep     insd                                    ; Repeat input to string
;;      rep     insw                                    ; Repeat input to string
;
;   /* Release an adapter buffer */
;   outport(bicd_base_addr + MSG_RAM_PTR, rbd_base_addr + 2 * curr_adap_rbd);
;   outport(bicd_base_addr + MSG_RAM, 0);
;   outport(bicd_base_addr + MSG_RAM, stats.config.buf_size);
;
        mov     edx, [ebp].IOMsgRamPtr                  ; Ram Ptr I/O port
        mov     eax, [ebp].CurrentAdapterRBD            ; EAX = adapter rbd
        shl     eax, 1                                  ; EAX = EAX * 2
        add     eax, RBD_BASE_ADDR                      ; EAX = EAX + base
        out     dx, ax                                  ; set ram ptr to adap rb
d

        mov     edx, [ebp].IOMsgRam                     ; Ram I/O port
        xor     eax, eax                                ; Clear status for reuse
        out     dx, ax

        mov     eax, RBD_BUFFER_SIZE                    ; buffer size
        out     dx, ax                                  ; set adapters buf size
;
;   /* Bump Current Adapter RBD value and make sure it wraps */
;   if (++curr_adap_rbd == stats.cnofig.adap_rbd_num)
;      curr_adap_rbd = 0;
;
        mov     eax, [ebp].CurrentAdapterRBD            ; EAX = current adap rbd
        inc     eax                                     ; Bump it
        cmp     eax, ADAP_RBD_NUM                       ; Did it go over?
        jb      DriverISRRBDWrap                        ; Jump if not
        xor     eax, eax                                ; Force it to wrap
DriverISRRBDWrap:
        mov     [ebp].CurrentAdapterRBD, eax            ; Save for later
        DebugMessage1   DEBUG_ISR_ALL, AdapterRBDMsg, eax
;
```

```
;   /* Now set ram ptr to next rbd in case we loop up to DriverISRLoop */
;   outport(bicd_base_addr + MSG_RAM_PTR, rbd_base_addr + 2*curr_adap_rbd);
;
        mov     edx, [ebp].IOMsgRamPtr
        shl     eax, 1
        add     eax, RBD_BASE_ADDR
        out     dx, ax
;
;   /* This was the first buffer. Save it off for unravel */
;      first_buffer_rbd = curr_rbd;
;      first_buffer = 0;
;      frame_buffers = 0;
;   }
;

;   if (!(status & EOF_BIT))
;   {
;
        test    [ebp].IntStatus, EOF_BIT                ; Last buffer?
        je      DriverISRLoop                           ; Jump if it wasn't
;
;   /* We have the last frame. Pass buffers to application. */
;   /*
;    * First lets see if we can match the packet address to a
;    * filter table address.
;    */
;   for (ii = 0; ii < BICDD_MAX_ADDR; ii++)
;    if (!fmemcmp( filter[ii].address, global_pool[first_buffer_rbd].buf_ptr, 6
)
;     && (filter[ii].channel != BICDD_NOT_USED)
;
        mov     [ebp].CurrentECB, 0                     ; Clear ECB flag
        lea     edi, [ebp].Filter                       ; EDI -> filter[0]
DriverISRFilterLoop:
        mov     eax, dword ptr [edi].FilterAddress+0    ; EAX = 1st filter addr
dword
        mov     edx, [esi].RPacketOffset                ; EDX -> buffer
        cmp     eax, [edx+0]                            ; 1st dword compare?
        jne     DriverISRFilterNext                     ; Jump if not
        mov     ax, word ptr [edi].FilterAddress+4      ; AX = last filter addr
word
        cmp     ax, [edx+4]                             ; last word compare?
        jne     DriverISRFilterNext                     ; Jump if not
;
; /* We have a winner! */
; /* EDI points to Filter entry */
;
        mov     eax, [edi].FilterChannel                ; EAX = channel
        cmp     eax, RBD_NOT_USED                       ; Valid channel?
        je      DriverISRFilterNext                     ; Jump if not

        DebugMessage6   DEBUG_ISR_ALL, FilterAddrMsg, [edx+0], [edx+1], [edx+2],
[edx+3], [edx+4], [edx+5]
;
; /* Lets set ESI to Rx Control entry */
;      cc = filter[ii].channel;
;
        lea     ebx, [ebp].RxControl[eax*8]             ; EBX -> Rx Control entr
y
;
;      /* Check whether length in the header matches the length rxed */
;      for (ll = 0, length = 0, mm = first_buffer-rbd; ll < frame_buffers; ll++
)
;      {
;
```

```
;       /* ECX = length = total length of all buffers */
;       p_length = (global_pool[first_buffer_rbd].buf_ptr) + 3;
;       if (((*p_length + 12) > length) || ((*p_length + 12) < (length - 16)))
;       {
;
        mov     ecx, [esi].RPacketSize                  ; ECX = hardware size
        mov     edx, [esi].RPacketOffset                ; EDX -> packet
        mov     edx, [edx+6]                            ; EDX = header length
        and     edx, 0ffffh                             ; we only need the word
        add     edx, 12                                 ; EDX = header length +
12
        cmp     edx, ecx                                ; *p_length+12 > length
        ja      DriverISRFilterSkipRBDsLen              ; Jump if it is
        sub     ecx, 16                                 ; ECX = length - 16
        cmp     edx, ecx                                ; *p_length+12 < length-
16
        jb      DriverISRFilterSkipRBDsLen              ; Jump if it is
;
;       filter[ii].total_count++;
;
        inc     [edi].FilterTotalCount                  ; Bump filter total coun
t
;
;       /* Check address seq numbers and update stats
;       p_seq_num = (global_pool[first_buffer_rbd].buf_ptr) + 2;
;
        mov     edx, [esi].RPacketOffset                ; EDX -> packet
        mov     eax, [edx+8]                            ; EAX = sequence number
;
;       if (*p_seq_num && filter[ii].seq_num && (*p_seq_num != filter[ii].seq_nu
m++)
;       {
;          filter[ii].seq_count++;
;          filter[ii].seq_num = *p_seq_num + 1;
;       }
;       else if (!*p_seq_num)
;          filter[ii].seq_num = 1;
;       else if (!filter[ii].seq_num)
;          filter[ii].seq = *p_seq_num + 1;
;
        or      eax, eax                                ; did header have a sequ
ence?
        je      DriverISRFilterNoPacketSeq              ; jump if not
        cmp     [edi].FilterSeqNum, 0                   ; does filter sequence e
xist?
        je      DriverISRFilterNoFilterSeq              ; jump if not
        cmp     eax, [edi].FilterSeqNum                 ; Do they match?
        jne     DriverISRFilterSeqNoMatch               ; Jump if not.
        inc     [edi].FilterSeqNum                      ; filter[ii].seq_num++
;
;       /* Discard frames if it's a "stats only" channel */
;       if (rx_cntl[cc].flags & BICDD_FLAGS_STATS_ONLY)
;       {
;
DriverISRFilterCallESR:
        mov     eax, [ebx].RxESR                        ; EAX -> channel ESR
        or      eax, eax
        je      DriverISRDidntWantECB                   ; Jump if no ESR
        cmp     eax, 0ffffffffh
        jne     DriverISRNotOurs

        public DriverISRInternet
DriverISRInternet:
;       INT_3
        lea     edi, [esi+RPacketEnvelope]              ; EDI -> EII header
        mov     ebx, [ebp].MSMDefaultVirtualBoard
```

```
        mov     ecx, [esi].RPacketOffset                ; ECX -> Satelite header
        ;
        ; Make sure its not a data feed address. We don't know what to do
        ; with these yet.
        ;
        mov     al, [ecx+0]
        mov     ah, [ecx+2]
        and     al, 3
        cmp     al, 3
        je      MightBeDataStreamPacket
        and     al, 1
        cmp     al, 1
        jne     NotDataStreamPacket
MightBeDataStreamPacket:
        cmp     ah, 0ffh
        jne     NotDataStreamPacket
        ;
        ; Data stream, go into debugger.
        ;
        INT_3                                           ; Break into debugg
        MSMReturnRCB                                    ; Return it
        jmp     DriverISRLoop                           ; Get next packet


NotDataStreamPacket:
        ;
        ; Fill in EII Destination with our node address
        ;
        mov     eax, dword ptr [ebx].MLIDNodeAddress
        mov     [edi+0], eax
        mov     ax, word ptr [ebx].MLIDNodeAddress+4
        mov     [edi+4], ax
        ;
        ; Fill in EII Source Address with (0x0a 0x0a 0x0a 0x0a 0x0a 0x0a)
        ;
;       mov     word ptr [edi+6], 0a0ah
;       mov     dword ptr [edi+8], 0a0a0a0ah
        mov     word ptr [edi+6], 0606h
        mov     dword ptr [edi+8], 06060606h
        ;
        ; Fill in EII type field with IP type (0x08 0x00)
        ;
        mov     word ptr [edi+12], 0008         ; Force IP PID(800h hi/lo) since
 DPC is unaware of it

        lea     ecx, [esi + RFragmentCount]
        push    ecx
        call    [DPCRxFrame]
        add     esp, 4

        movzx   ecx, word ptr [esi + RPacketEnvelope + 14 + 2]
        xchg    ch, cl
        add     ecx, 14
        cmp     ecx, 3ch
        jae     RxPacketIsPadded
        mov     ecx, 3ch
RxPacketIsPadded:

;
; vvv DEBUG
        mov     eax, ecx
        cmp     eax, 400
        jb      DebugRxExit

        cmp     eax, [ebp].LargestRx
        jbe     DebugRxAve
        mov     [ebp].LargestRx, eax

DebugRxAve:
        inc     [ebp].NumberLargeRx
        add     eax, [ebp].TotalLargeRx
        mov     edi, [ebp].NumberLargeRx
        mov     [ebp].TotalLargeRx, eax
        xor     edx, edx
        div     edi
        mov     [ebp].AveLargeRx, eax

DebugRxExit:

; ^^^ DEBUG
;
        mov     edi, 1514                       ; Max Packet size
if TIMESTAMP
;       push    ecx
;       mov     al, 'R'
;       push    eax
;       call    DPCTimestamp
;       lea     esp, [esp + 4]
;       pop     ecx
endif
        xor     eax, eax                        ; Good packet

        push    ebp
        call    EtherTSMFastProcessGetRCB
        pop     ebp
        jne     DriverISRLoop                                   ; Jump i
f no ecb returned
        jmp     DriverISRDidntWantECB           ; Give it back. We don't store t
hese

DriverISRNotOurs:
        push    esi                             ; Parm0 = ECB
        call    eax                             ; call ESR
        pop     esi                             ; clean up stack
        or      eax, eax                        ; did they keep it?
        je      DriverISRLoop                   ; jump if they did.

DriverISRDidntWantECB:
        MSMReturnRCB                            ; Return it
        jmp     DriverISRLoop                   ; Get next packet


DriverISRFilterNext:
        add     edi, size FilterStruct
        lea     eax, [ebp].Filter[MAX_ADDR * size FilterStruct]
        cmp     edi, eax
        jbe     DriverISRFilterLoop
;
; Couldn't find filter address. Clean up.
;

        MSMReturnRCB                            ; Give ECB back.
        DebugMessage6   DEBUG_ISR_ALL, FilterNone, [edx+0], [edx+1], [edx+2], [e
dx+3], [edx+4], [edx+5]
        jmp     DriverISRLoop


DriverISRFilterSeqNoMatch:
        inc     eax
        inc     [edi].FilterSeqCount
        mov     [edi].FilterSeqNum, eax
        jmp     DriverISRFilterCallESR
```

```

DriverISRFilterNoFilterSeq:
        inc     eax
        mov     [edi].FilterSeqNum, eax
        jmp     DriverISRFilterCallESR

DriverISRFilterNoPacketSeq:
        mov     [edi].FilterSeqNum, 1
        jmp     DriverISRFilterCallESR

DriverISRFilterSkipRBDsLen:
        MSMReturnRCB
        DebugMessage2   DEBUG_ISR_ALL, FilterRBDLen, ecx, edx
        jmp     DriverISRLoop

DriverISRNoECB:
        DebugMessage    DEBUG_ISR, NoECBMsg
        jmp     DriverISRBadNextRBD

DriverISRBadPacket:
        mov     esi, [ebp].IntStatus
        test    esi, FRAMING_ERR
        je      DriverISRCheckAbort

        DebugMessage    DEBUG_ISR, FramingErrMsg

DriverISRCheckAbort:
        test    esi, ABORT
        je      DriverISRCheckAlign

        DebugMessage    DEBUG_ISR, AbortMsg

DriverISRCheckAlign:
        test    esi, ALIGN_ERR
        je      DriverISRCheckOverrun

        DebugMessage    DEBUG_ISR, AlignErrMsg

DriverISRCheckOverrun:
        test    esi, OVERRUN_ERR
        je      DriverISRCheckDES

        DebugMessage    DEBUG_ISR, OverrunErrMsg

DriverISRCheckDES:
        test    esi, DES_ERR
        je      DriverISRCheckCRC

        DebugMessage    DEBUG_ISR, DESErrMsg

DriverISRCheckCRC:
        test    esi, CRC_ERR
        je      DriverISRErrorStats

        DebugMessage    DEBUG_ISR, CRCErrMsg

DriverISRErrorStats:

        DebugMessage    DEBUG_ISR, ReturnMsg

DriverISRBadNextRBD:
        mov     edx, [ebp].IOMsgRamPtr
        mov     eax, [ebp].CurrentAdapterRBD
        shl     eax, 1
        add     eax, RBD_BASE_ADDR
        out     dx, ax
        mov     edx, [ebp].IOMsgRam
        xor     eax, eax
        out     dx, ax

        mov     eax, RBD_BUFFER_SIZE
        out     dx, ax

        mov     eax, [ebp].CurrentAdapterRBD
        inc     eax
        cmp     eax, ADAP_RBD_NUM
        jb      DriverISRBadRBDWrap
        xor     eax, eax
DriverISRBadRBDWrap:
        mov     [ebp].CurrentAdapterRBD, eax

        DebugMessage1   DEBUG_ISR_ALL, AdapterRBDMsg, eax
        mov     edx, [ebp].IOMsgRamPtr
        shl     eax, 1
        add     eax, RBD_BASE_ADDR
        out     dx, ax
        jmp     DriverISRLoop

DriverISRExit:
        mov     edx, [ebp].IOMsgRamPtr
        mov     eax, 0c3a0h
        out     dx, ax

        mov     eax, [ebp].CurrentAdapterRBD
        mov     edx, [ebp].IOMsgRam
        out     dx, ax

        DebugMessage1   DEBUG_ISR_ALL, ISRExitMsg, eax

        mov     edx, [ebp].IOStatus
        in      ax, dx

        ret

DriverISR       endp
        subttl  -- DriverDisableInterrupt --
        page
;
;*****************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverDisableInterrupt, DPC/API/DISINT )
;
; Name:         DriverDisableInterrupt
;
; Description:  This routine will disable the adapters ability to
;               interrupt the host.
;
; On Entry:     EAX     N/A
;               EBX     N/A
;               ECX     N/A
;               EDX     N/A
;               EBP     @ Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are disabled.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Preserved
;               EDX     Destroyed
```

```
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts disabled.
;
; Remarks:      This routine is called by the MSM.
;
; See Also:     DriverEnableInterrupt
;
; END_MANUAL_ENTRY
;
;*****************************************************************/
;
        align   16
DriverDisableInterrupt  proc

        xor     eax, eax
        ret

DriverDisableInterrupt  endp
        subttl  -- DriverEnableInterrupt --
        page
;
;*****************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverEnableInterrupt, DPC/API/ENINT )
;
; Name:         DriverEnableInterrupt
;
; Description:  This routine will enable the adapters ability to
;               interrupt the host.
;
; On Entry:     EAX     N/A
;               EBX     N/A
;               ECX     N/A
;               EDX     N/A
;               EBP     @ Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are disabled.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Preserved
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts disabled.
;
; Remarks:      This routine is called by the MSM.
;
; See Also:     DriverDisableInterrupt
;
; END_MANUAL_ENTRY
;
;*****************************************************************/
;
        align   16
```

```
DriverEnableInterrupt   proc

        ret

DriverEnableInterrupt   endp
        public  DriverReset
        subttl  -- DriverReset --
        page
;
;*****************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverReset, DPC/API/RESET )
;
; Name:         DriverReset
;
; Description:  This routine will reset and initialize the NIC.
;
; On Entry:     EAX     N/A
;               EBX     @ Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     @ Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are disabled.
;
; On Return:    EAX     0 if successful(otherwise points to error message)
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Destroyed
;               EDI     Destroyed
;
;               Flags:
;
;               Note:   Interrupts disabled.
;
; Remarks:      This routine is called by the MSM media module.
;               It is called at process time.
;
; See Also:     ETHERTSM\EtherTSMReset
;
; END_MANUAL_ENTRY
;
;*****************************************************************/
;
DriverReset     proc    near

        inc     [ebp].AdapterResetCount         ; Increment stat counter.

        xor     eax, eax
        ret

DriverReset     endp

DefaultRxFrame  proc

        ret

DefaultRxFrame  endp

        extrn   LSLGetStackIDFromName: near
ProtocolBindEvent       proc
```

```
        CPush

        lea     edx, IPName
        call    LSLGetStackIDFromName   ; Return Stack ID in EBX
        or      eax, eax
        jne     short ProtocolBindExit

        mov     esi, [esp + Parm0]
        cmp     [esi+4], ebx                    ; IP Stack?
        jne     short ProtocolBindExit  ; Nope
        mov     edx, [esi]                      ; EDX = Bound board number
        mov     ebp, OurAdapterDataSpace

        xor     ecx, ecx
ProtocolBindLoop:
        mov     ebx, [ebp+MSMVirtualBoardLink][ecx*4]
        or      ebx, ebx
        jz      ProtocolBindNext

        cmp     [ebx].MLIDBoardNumber, dx
        jne     ProtocolBindNext

        mov     eax, 1514
        mov     [ebx].MLIDMaximumSize, eax
        sub     eax, 14
        mov     [ebx].MLIDMaxRecvSize, eax
        mov     [ebx].MLIDRecvSize, eax

ProtocolBindNext:
        inc     ecx
        cmp     ecx, 4
        jb      ProtocolBindLoop
ProtocolBindExit:
        CPop
        ret

ProtocolBindEvent       endp

ProtocolUnbindEvent     proc

        CPush

        lea     edx, IPName
        call    LSLGetStackIDFromName   ; Return Stack ID in EBX
        or      eax, eax
        jne     short ProtocolUnbindExit

        mov     esi, [esp + Parm0]
        cmp     [esi+4], ebx                    ; IP Stack?
        jne     short ProtocolBindExit  ; Nope
        mov     edx, [esi]                      ; EDX = Bound Board Number
        mov     ebp, OurAdapterDataSpace

        xor     ecx, ecx
ProtocolUnbindLoop:
        mov     ebx, [ebp+MSMVirtualBoardLink][ecx*4]
        or      ebx, ebx
        jz      ProtocolUnbindNext

        cmp     [ebx].MLIDBoardNumber, dx
        jne     ProtocolUnbindNext

        mov     eax, 1494
        mov     [ebx].MLIDMaximumSize, eax
        sub     eax, 14
        mov     [ebx].MLIDMaxRecvSize, eax
        mov     [ebx].MLIDRecvSize, eax

ProtocolUnbindNext:
        inc     ecx
        cmp     ecx, 4
        jb      ProtocolUnbindLoop
ProtocolUnbindExit:
        CPop
        ret

ProtocolUnbindEvent     endp


        subttl  -- DriverInit --
        page
;
;*********************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverInit, DPC/API/INIT )
;
; Name:         DriverInit
;
; Description:  This routine will call EtherTSMRegisterHSM,
;               MSMParseDriverParameters, MSMRegisterHardwareOptions,
;               MSMSetHardwareInterrupt, MSMRegisterMLID, initialize
;               variables in the Adapter Data Space and reset/initialize
;               the card.
;
; On Entry:     EAX     N/A
;               EBX     N/A
;               ECX     N/A
;               EDX     N/A
;               EBP     N/A
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are enabled.
;
; On Return:    EAX     0 if successful(otherwise it points to error message)
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by the OS at load time.
;               It is called at process time.
;
; See Also:     MSM\MSMParseDriverParameters
;               MSM\MSMRegisterHardwareOptions
;               MSM\MSMSetHardwareInterrupts
;               MSM\MSMRegisterMLID
;               MSM\MSMScheduleIntTimeCallBack
;               MSM\MSMScheduleAESCallBack
;               MSM\MSMEnablePolling
;               DriverReset
;
; END_MANUAL_ENTRY
;
;*********************************************************************/
;
```

```
        extrn   RegisterForEventNotification: near
        extrn   UnRegisterEventNotification: near

DriverInit      proc

        CPush
if TIMESTAMP
        lea     eax, DPCTB
        mov     timestamp_begin, eax
        mov     timestamp_index, eax
        add     eax, TIMESTAMP_BUFFER_SIZE
        mov     timestamp_end, eax
endif
;
;*****************************************************************\
;                                                                 *
; Fill in Driver Parameter Block fields.                          *
;                                                                 *
;*****************************************************************/
;
        mov     DriverStackPointer, esp                 ; Fill in stack ->.

        lea     esi, DriverParameterBlock               ; ESI -> Parm block.
        call    EtherTSMRegisterHSM                     ; Get EBX.
        jnz     DriverInitError                         ; Jump if error.
;
; Yuck! We'll have to adjust the receive size down, since
; Hughes can't handle full 1500 byte packets with tunneling.
;
        mov     [ebx].MLIDMaximumSize, 1494
;
;*****************************************************************\
;                                                                 *
; EBX -> Frame Data Space(Config Table).                          *
; Let MSM Parse the command line.                                 *
;                                                                 *
;*****************************************************************/
;
        mov     GlobalRxFreq, DEFAULT_RX_FREQ

        mov     eax, NeedsIOPort0Bit OR NeedsInterrupt0Bit OR CAN_SET_NODE_ADDRE
SS
        lea     ecx, AdapterOptions
        call    MSMParseDriverParameters
        jnz     DriverInitError                         ; Jump if error.
;
;*****************************************************************\
;                                                                 *
; Let MSM Register the hardware options.                          *
;                                                                 *
;*****************************************************************/
;
        call    MSMRegisterHardwareOptions
        cmp     eax, 1                                  ; Error Registering?
        ja      DriverInitError                         ; Jump if so.
        je      DriverInitExit                          ; Skip if new frame.

        mov     OurAdapterDataSpace, ebp                        ; Save for later
        mov     DPCRxFrame, offset DefaultRxFrame
;
; Get a timer resource tag so that we can delay ourselves.
;
        push    TimerSignature
        push    offset TimerDesc
      \ push    DriverModuleHandle
        call    AllocateResourceTag
        lea     esp, [esp + (3 * 4)]
        mov     [ebp].TimerTag, eax
        or      eax, eax
        lea     eax, ErrorAllocatingRTagMessage
        je      DriverInitErrorReturn
;
; Get a resource tag for interrupts.
;
        push    InterruptSignature                  ; Pass signature.
        push    offset InterruptRTagMessage         ; Pass Message.
        push    DriverModuleHandle                  ; Pass Module handle.
        call    AllocateResourceTag                 ; Allocate RTag.
        add     esp, (3 * 4)                        ; Clean up stack.
        mov     [ebp].ISRTag, eax                   ; Save RTag.
        or      eax, eax                            ; RTag returned?
        lea     eax, ErrorAllocatingRTagMessage ; EAX -> Error mess.
        je      DriverInitErrorReturn           ; Jump if not.
;
; Get a resource tag for event notification.
;
        push    EventSignature                  ; Pass signature.
        push    offset EventRTagMessage         ; Pass Message.
        push    DriverModuleHandle                  ; Pass Module handle.
        call    AllocateResourceTag                 ; Allocate RTag.
        add     esp, (3 * 4)                        ; Clean up stack.
        mov     [ebp].EventTag, eax                 ; Save RTag.
        or      eax, eax                            ; RTag returned?
        lea     eax, ErrorAllocatingRTagMessage ; EAX -> Error mess.
        je      DriverInitErrorReturn           ; Jump if not.
;
; Register for protocol bind event.
;
        mov     eax, [ebp].EventTag
        push    offset ProtocolBindEvent            ; Pass procedure.
        push    0                                   ; No Warning proc.
        push    EVENT_PRIORITY_OS                   ; Priority level.
        push    EVENT_PROTOCOL_BIND                 ; Event type.
        push    eax                                 ; Pass resource tag.
        call    RegisterForEventNotification    ; Register Event.
        add     esp, (4 * 5)
        mov     [ebp].ProtocolBindID, eax                   ; Save Event ID.
        or      eax, eax                            ; Registered?
        je      DriverInitNoBindEvent           ; Jump if not.

        mov     eax, [ebp].EventTag
        push    offset ProtocolUnbindEvent          ; Pass procedure.
        push    0                                   ; No Warning proc.
        push    EVENT_PRIORITY_OS                   ; Priority level.
        push    EVENT_PROTOCOL_UNBIND           ; Event type.
        push    eax                                 ; Pass resource tag.
        call    RegisterForEventNotification    ; Register Event.
        add     esp, (4 * 5)
        mov     [ebp].ProtocolUnbindID, eax         ; Save Event ID.
        or      eax, eax                            ; Registered?
        jne     DriverInitSetPorts                  ; Jump if not.

DriverInitNoBindEvent:
        mov     eax, 1494                           ; Just take the max down
        mov     [ebx].MLIDMaximumSize, eax
        sub     eax, 14
        mov     [ebx].MLIDMaxRecvSize, eax
        mov     [ebx].MLIDRecvSize, eax

DriverInitSetPorts:
;
;**********************************************************\
```

```
;                                                               *
; Set and check the adapters base I/O.                          *
;                                                               *
;***************************************************************/
;
                movzx   ecx, [ebx].MLIDIOPortsAndLengths
                mov     [ebp].IORxData, ecx                     ; Rx Data port = base +
0
                add     ecx, 2
                mov     [ebp].IOAutoInc, ecx                    ; Auto Inc port = base +
 2
                add     ecx, 2
                mov     [ebp].IOStatus, ecx                     ; Status port = base + 4
                add     ecx, 2
                mov     [ebp].IOControl, ecx                    ; Control port = base +
6
                add     ecx, 2
                mov     [ebp].IOMsgRamPtr, ecx                  ; Msg Ram Ptr = base + 8
                add     ecx, 2
                mov     [ebp].IOMsgRam, ecx                     ; Msg Ram = base +
                add     ecx, 2
                mov     [ebp].IORbdBufLen, ecx                  ; RDB buf Len = base + 1
2
                add     ecx, 2
                mov     [ebp].IORbdNum, ecx                     ; RDB Num = base + 14
                add     ecx, 2
                mov     [ebp].IOBtrControlAddr, ecx             ; BTR Control Addr = bas
e + 10h + 0h
                add     ecx, 2
                mov     [ebp].IOAfcControlAddr, ecx             ; AFC Control Addr = bas
e + 10h + 2h
                add     ecx, 2
                mov     [ebp].IOBitDetControlAddr, ecx          ; Bit Det Control Addr =
 base + 10h + 4h
                add     ecx, 2
                mov     [ebp].IOAgcFirControlAddr, ecx          ; Agc Fir Control Addr =
 base + 10h + 6h
                add     ecx, 2
                mov     [ebp].IOCrlkThrLowAddr, ecx             ; Crlk Thr Low Addr = ba
se + 10h + 8h
                add     ecx, 2
                mov     [ebp].IOCthAddr, ecx                    ; Cth Addr = base + 10h
+ 0ah
                add     ecx, 2
                mov     [ebp].IOGateCountHighAddr, ecx          ; Gate Count High Add
 base + 10h + 0ch
                add     ecx, 2
                mov     [ebp].IOCountNomLowAddr, ecx            ; Count Nom Low Addr = b
ase + 10h + 0eh
                add     ecx, 2
                mov     [ebp].IOCountNomHighAddr, ecx           ; Count Nom High Addr =
base + 10h + 10h
                add     ecx, 2
                mov     [ebp].IOCountDeltaAddr, ecx             ; Count Delta Addr = bas
e + 10h + 12h
                add     ecx, 2
                mov     [ebp].IOSweepRateAddr, ecx              ; Sweep Rate Addr = base
 + 10h + 14h
                add     ecx, 2
                mov     [ebp].IOCrlkControlAddr, ecx            ; Crlk Control Addr = ba
se + 10h + 16h
                add     ecx, 2
                mov     [ebp].IOSynthSerControlAddr, ecx        ; Synth Ser Control Addr
 = base + 10h + 18h
                add     ecx, 2
                mov     [ebp].IOSpareIOControlAddr, ecx         ; Spare IO Control Addr
```

```
= base + 10h + 1ah
        add     ecx, 2
        mov     [ebp].IODaAdOffsetControlAddr, ecx      ; IO DA Offset Control A
ddr = base + 10h + 1ch
        add     ecx, 2
        mov     [ebp].IOUnitIDAddr, ecx                 ; Unit ID Addr = base +
10h + 1eh
        add     ecx, 2

        movzx   ecx, [ebx].MLIDIOPortsAndLengths
        mov     al, 0bh                                 ; AL = 0bh
        cmp     ecx, 100h                               ; I/O port 100h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 0ah
        cmp     ecx, 140h                               ; I/O port 140h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 9
        cmp     ecx, 180h                               ; I/O port 180h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 8
        cmp     ecx, 1c0h                               ; I/O port 1c0h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 7
        cmp     ecx, 200h                               ; I/O port 200h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 6
        cmp     ecx, 240h                               ; I/O port 240h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 5
        cmp     ecx, 280h                               ; I/O port 280h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 4
        cmp     ecx, 2c0h                               ; I/O port 2c0h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 3
        cmp     ecx, 300h                               ; I/O port 300h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 2
        cmp     ecx, 340h                               ; I/O port 340h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 1
        cmp     ecx, 380h                               ; I/O port 380h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 0
SetPort:
        mov     dx, 279h
        out     dx, al                                  ; Set Base I/O port
;
; Lets Reset the adapter.
;
        push    eax

        mov     edx, [ebp].IOControl
        mov     eax, CNTL_MRESET
        out     dx, ax

        mov     eax, [ebp].TimerTag
        push    eax
        push    2
        call    DelayMyself
        add     esp, (2 * 4)

        mov     edx, [ebp].IOControl
        mov     eax, 0
        out     dx, ax
```

```
        pop     eax
        mov     dx, 279h
        out     dx, al
;
; Make sure that we can set spare output
;
        mov     edx, [ebp].IOControl                    ; Set Spare Output
        mov     eax, CNTL_SOUTPUT
        out     dx, ax

        mov     edx, [ebp].IOStatus
        in      ax, dx
        test    eax, STAT_SOUTPUT
        mov     eax, offset MsgIOSetFailed
        je      DriverInitErrorReturn
;
; Make sure that we can clear spare output
;
        mov     edx, [ebp].IOControl                    ; Clear Spare Outpu
        mov     eax, 0
        out     dx, ax

        mov     edx, [ebp].IOStatus
        in      ax, dx
        test    eax, STAT_SOUTPUT
        mov     eax, offset MsgIOClearFailed
        jne     DriverInitErrorReturn
;
; If no node override, default it.
;
        cmp     dword ptr [ebx].MLIDNodeAddress, -1
        jnz     short NodeIsSet
        mov     [ebx].MLIDNodeAddress+0, 00h
        mov     [ebx].MLIDNodeAddress+1, 80h
        mov     [ebx].MLIDNodeAddress+2, 0aeh
        mov     [ebx].MLIDNodeAddress+3, 00h
        mov     [ebx].MLIDNodeAddress+4, 00h
        mov     [ebx].MLIDNodeAddress+5, 01h

NodeIsSet:
;
;*****************************************************************\
;                                                                 *
; Download the MIPS code to the adatper.                          *
;                                                                 *
;*****************************************************************/
;
        mov     edx, [ebp].IOControl            ; Set to auto-increment mode
        mov     eax, CNTL_AUTO_INC
        out     dx, ax

        mov     edx, [ebp].IOMsgRamPtr          ; Set Adapter Ram Address
        xor     eax, eax                        ;  to zero
        out     dx, ax

        mov     edx, [ebp].IOMsgRamPtr          ; Set Adapter Ram Address
        xor     eax, eax                        ;  to zero
        out     dx, ax

        mov     ecx, MipsCodeSize               ; Number of words to copy
        mov     edx, [ebp].IOMsgRam             ; adapter ram port
        lea     esi, MipsCode                   ; ESI -> mips code
        cld
CopyToAdapterLoop:
        lodsw                                   ; Get Mips Word
        out     dx, ax                          ; Send it to adapter
        dec     ecx
        jnz     CopyToAdapterLoop
;
; Verify the download by reading the data back
;
        mov     edx, [ebp].IOControl            ; Set to auto-increment mode
        mov     eax, CNTL_AUTO_INC
        out     dx, ax

        mov     edx, [ebp].IOMsgRamPtr          ; Set Adapter Ram Address
        xor     eax, eax                        ;  to zero
        out     dx, ax

        mov     ecx, MipsCodeSize               ; Number of words to copy
        mov     edx, [ebp].IOMsgRam             ; adapter ram port
        lea     esi, MipsCode                   ; ESI -> mips code
        cld
VerifyAdapterLoop:
        xor     eax, eax
        lodsw                                   ; Get Mips Word
        mov     edi, eax
        in      ax, dx                          ; Send it to adapter
        cmp     edi, eax
        lea     eax, MsgBadRAM
        jne     DriverInitErrorReturn
        dec     ecx
        jnz     VerifyAdapterLoop
;
;***********************************************************\
;                                                           *
; Register our interrupt handler with the OS.               *
;                                                           *
;***********************************************************/
;
        mov     edx, [ebp].IOStatus             ; Clear old errors
        in      ax, dx
;
; Set RBD base address
;
        mov     edx, [ebp].IORbdBase
        mov     eax, RBD_BASE_ADDR
        out     dx, ax
        mov     edx, [ebp].IOMsgRamPtr
        out     dx, ax

        mov     edx, [ebp].IORbdNum
        mov     eax, ADAP_RBD_NUM
        out     dx, ax
        mov     ecx, eax

        mov     edx, [ebp].IORbdBufLen
        mov     eax, RBD_BUFFER_SIZE
        out     dx, ax

        mov     edx, [ebp].IOControl
        mov     eax, CNTL_AUTO_INC
        out     dx, ax

        mov     edx, [ebp].IOMsgRam
SetupBuffersLoop:
        xor     eax, eax
        out     dx, ax

        mov     eax, RBD_BUFFER_SIZE
        out     dx, ax
```

```
        dec     ecx
        jnz     SetupBuffersLoop
;
; Enable the adapter.
;
        movzx   ecx, [ebx].MLIDInterrupt
        mov     esi, CNTL_IRQ3
        mov     edx, 8
        cmp     ecx, 8h
        je      EnableDPC
        mov     esi, CNTL_IRQ4
        mov     edx, 10h
        cmp     ecx, 4
        je      EnableDPC
        mov     esi, CNTL_IRQ5
        mov     edx, 20h
        cmp     ecx, 5
        je      EnableDPC
        mov     esi, CNTL_IRQ9
        mov     edx, 2h
        cmp     ecx, 8
        je      EnableDPC
        mov     esi, CNTL_IRQ10
        mov     edx, 4h
        cmp     ecx, 10
        je      EnableDPC
        mov     esi, CNTL_IRQ11
        mov     edx, 8h
        cmp     ecx, 11
        je      EnableDPC
        mov     esi, CNTL_IRQ12
        mov     edx, 10h
        cmp     ecx, 12
        je      EnableDPC
        mov     esi, CNTL_IRQ15
        mov     edx, 80h
EnableDPC:
        mov     [ebp].PicMask, edx
        not     edx
        mov     [ebp].PicUnMask, edx
        mov     [ebp].PicAddress, 21h
        cmp     [ebx].MLIDInterrupt, 8
        jb      ClearOurInterrupt
        mov     [ebp].PicAddress, 0a1h
ClearOurInterrupt:
        mov     edx, [ebp].PicAddress
        in      al, dx
        and     eax, [ebp].PicUnMask
        out     dx, al

        mov     eax, esi
        mov     edx, [ebp].IOControl
        or      eax, CNTL_RX_EN OR CNTL_CPU_EN OR CNTL_INT_EN OR CNTL_SNGL_INT O
R CNTL_AUTO_INC OR CNTL_SOUTPUT
        out     dx, ax
        mov     [ebp].IOEnableValue, eax

        cmp     DebugMask, 0
        je      OpenScreenExit
        push    4e524353h                       ; 'NRCS'
        lea     eax, ScreenResourceName
        push    eax
        push    DriverModuleHandle
        call    AllocateResourceTag
        lea     esp, [esp + (3 * 4)]
        or      eax, eax
        je      OpenScreenExit

        mov     ScreenRTag, eax

        push    offset DPCScreen                ; Location of Handle storage
        push    eax                             ; Screen Resource Tag
        push    offset ScreenName               ; Name of screen object
        call    OpenScreen
        lea     esp, [esp + (3 * 4)]
        or      eax, eax
        jne     OpenScreenExit

        push    offset NLMName
        push    0
        push    DriverModuleHandle
        call    GetNLMNames
        lea     esp, [esp + (3 * 4)]

        push    0
        push    offset Day
        push    offset Month
        push    offset Year
        push    0
        push    offset MinorVer
        push    offset MajorVer
        push    DriverModuleHandle
        call    GetNLMVersionInfo
        lea     esp, [esp + (8 * 4)]

        push    Year
        push    Day
        push    Month
        push    MinorVer
        push    MajorVer
        push    offset NLMName
        push    offset DebugScreenHeader
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (8 * 4)]

OpenScreenExit:
;
;**********************************************************\
;                                                         *
; Initialize RxControl and Filter entries.                *
;                                                         *
;**********************************************************/
;
        lea     edi, [ebp].RxControl
        mov     ecx, MAX_CHAN
        xor     eax, eax
        mov     edx, RBD_NOT_USED
InitRxControlLoop:
        mov     [edi].RxChannel, edx
        mov     [edi].RxESR, eax
        add     edi, size RX_CNTL
        dec     ecx
        jne     InitRxControlLoop

        lea     edi, [ebp].Filter
        mov     ecx, MAX_ADDR
InitFilterLoop:
        mov     [edi].FilterChannel, edx
        mov     [edi].FilterTotalCount, eax
        mov     [edi].FilterSeqCount, eax
        mov     [edi].FilterSeqNum, eax
```

```
        add     edi, size FilterStruct
        dec     ecx
        jne     InitFilterLoop
;
;**********************************************************\
;
; Test Interrupts.
;
;**********************************************************/
;

        ;
        ; Set ISR to test routine.
        ;
        mov     [ebp].GotInterrupt, 0   ; Clear test flag.
        mov     ecx, [ebp].ISRTag               ; ECX = ISR resource tag.
        push    0                               ; No ExtraEOIFlag offset.
        push    0                               ; ShareFlag.
        push    0                               ; End of chain flag.
        push    ecx                             ; ISR Resource tag.
        lea     eax, TestDriverISR
        push    eax                             ; ISR Entry point.
        movzx   eax, [ebx].MLIDInterrupt        ; EAX = Interrupt number.
        push    eax                             ; Interrupt Number.
        call    SetHardwareInterrupt            ; Set interrupt.
        lea     esp, [esp + (6 * 4)]            ; Restore stack.
        or      eax, eax                        ; Error setting interrupt?
        lea     eax, BadISRMsg          ; EAX -> Error Message.
        jnz     DriverInitErrorReturn   ; Exit if so.

        mov     edx, [ebp].IOControl
        mov     eax, [ebp].IOEnableValue
        or      eax, CNTL_FORCEINT
        out     dx, ax

        sti
        mov     eax, [ebp].TimerTag
        push    eax
        push    18
        call    DelayMyself
        add     esp, (2 * 4)
        cli

        movzx   eax, [ebx].MLIDInterrupt
        lea     edx, TestDriverISR
        push    edx                                     ; Pass ISR Entry poi
        push    eax                                     ; Pass Interrupt #.
        call    ClearHardwareInterrupt          ; Give interrupt back.
        lea     esp, [esp + (2 * 4)]                    ; Clean up stack.

        mov     eax, [ebp].IOEnableValue
        out     dx, ax

        lea     eax, NoInterruptMsg             ; Error message.
        cmp     [ebp].GotInterrupt, 0   ; Did we get it?
        je      DriverInitErrorReturn   ; Jump if not.
;
;**********************************************************\
;                                                         *
; EBX -> Frame Data Space(Config Table).                  *
; EBP -> Adapter Data Space.                              *
;                                                         *
; Let MSM Set Hardware Interrupt.                         *
;                                                         *
;**********************************************************/
;
        call    MSMSetHardwareInterrupt
        jnz     DriverInitError                 ; Jump if error.
;
;*************************************************************\
;                                                             *
; Set TxFreeCount to make TSM happy.                          *
;                                                             *
;*************************************************************/
;
        mov     [ebp].MSMTxFreeCount, 32                ; Allow 32 transmits sim
ultaneously.

        mov     eax, 1                          ; Schedule call back in 18 ticks

        call    MSMScheduleIntTimeCallBack
        jnz     DriverInitError         ; Jump if error.

        call    DriverReset                     ; Initialize NIC.
        jnz     DriverInitErrorReturn   ; Exit if error reseting.

        mov     [ebp].FirstTimeInit, 0          ; Disable DriverReset from
                                                ; testing the hardware again.

        dec     [ebp].AdapterResetCount         ; Adjust reset count.

        call    MSMRegisterMLID         ; Register MLID.
        jnz     DriverInitError         ; Jump if error.

        ; Lets see if the adapter is locked up.

        mov     eax, [ebp].TimerTag
        push    eax
        push    18
        call    DelayMyself
        add     esp, (2 * 4)

        call    RefreshMipsStats
        test    [ebp].MipsRxEnables, 8000000h   ; This shouldn't be big
        lea     eax, LockedAdapterMsg
        jne     DriverInitErrorReturn

        cmp     DebugMask, 0
        je      DriverInitExit
        movzx   eax, [ebx].MLIDInterrupt
        push    eax
        movzx   eax, [ebx].MLIDIOPortsAndLengths
        push    eax
        push    offset DebugInitOK
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (4 * 4)]

DriverInitExit:
        mov     [ebx].MLIDMaxRecvSize, 1400
        xor     eax, eax
        CPop
        ret

DriverInitErrorReturn:
        push    eax                             ; Save error message.
        call    MSMReturnDriverResources        ; Return resources.
        mov     eax, DPCScreen
        or      eax, eax
        je      DriverInitErrorScreenClosed
        push    eax
        call    CloseScreen
        lea     esp, [esp + (1 * 4)]
```

```
        mov     DPCScreen, 0
DriverInitErrorScreenClosed:
        pop     eax                             ; EAX -> Error message.

DriverInitError:
        mov     esi, eax                        ; ESI -> Error message.
        call    MSMPrintString                  ; Display message

        or      eax, 1                          ; Do not load return code.
        CPop

        ret

DriverInit      endp
        subttl  -- DriverShutdown --
        page
;
;*****************************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverShutdown, DPC/API/SHUTDOWN )
;
; Name:         DriverShutdown
;
; Description:  This routine will turn off the NIC.
;
; On Entry:     EAX     N/A
;               EBX     @ Frame Data Space
;               ECX     0 if Permanent Shutdown
;               EDX     N/A
;               EBP     @ Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are disabled.
;
; On Return:    EAX     0 if successful
;               EBX     Preserved
;               ECX     Preserved
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by the MSM media module.
;               It is called at process time.
;
; See Also:     ETHERTSM\EtherTSMShutdown
;
; END_MANUAL_ENTRY
;
;*****************************************************************************/
;
DriverShutdown  proc

        or      ecx, ecx
        jne     DriverShutdownAdapter
        mov     eax, [ebp].AgentRemoveRoutine
        or      eax, eax
        je      DriverShutdownAdapter

        call    eax
        mov     [ebp].AgentRemoveRoutine, 0

DriverShutdownAdapter:

        pushfd
        cli
        mov     edx, [ebp].IOControl
        xor     eax, eax
        out     dx, ax

        mov     edx, [ebp].IOStatus
        in      ax, dx

        or      ecx, ecx
        jne     DriverShutdownExit

        mov     edx, [ebp].IOControl
        mov     eax, CNTL_MRESET
        out     dx, ax

        mov     eax, DPCScreen
        or      eax, eax
        je      DriverShutdownScreenClosed
        push    eax
        call    CloseScreen
        lea     esp, [esp + (1 * 4)]
        mov     DPCScreen, 0
DriverShutdownScreenClosed:

        mov     eax, [ebp].ProtocolBindID
        or      eax, eax
        je      DriverShutdownExit
        push    eax                             ; Pass Event ID.
        call    UnRegisterEventNotification     ; Unregister event.
        add     esp, (1 * 4)                    ; Clean up stack.

        mov     eax, [ebp].ProtocolUnbindID
        or      eax, eax
        je      DriverShutdownExit
        push    eax                             ; Pass Event ID.
        call    UnRegisterEventNotification     ; Unregister event.
        add     esp, (1 * 4)                    ; Clean up stack.

DriverShutdownExit:
        popfd
        xor     eax, eax                        ; Good Return code.
        ret

DriverShutdown  endp
        subttl  -- DriverRemove --
        page
;
;*******************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverRemove, DPC/API/REMOVE )
;
; Name:         DriverRemove
;
; Description:  This routine call the MSM to return our resources.
;
; On Entry:     EAX     N/A
;               EBX     N/A
;               ECX     N/A
;               EDX     N/A
;               EBP     N/A
;               ESI     N/A
```

```
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by the OS at unload.
;               It is called at process time.
;
; See Also:     MSM\MSMDriverRemove
;
; END_MANUAL_ENTRY
;
;*******************************************************************/
;
DriverRemove    proc

        CPush
        mov     eax, DriverModuleHandle
        call    MSMDriverRemove
        CPop
        ret
;
DriverRemove    endp

OSCODE  ends

        end
```

```
/* interface between Helius DPCNE and Hughes DPCPE */

extern "C" {
#include <nwsemaph.h>
#include "sys_win.hhi"
#include "dpcutils.h"
#include "dbsinwin.h"
#undef VIRTUAL
#include "dpcagent.h"
#include <assert.h>

int PD_ESR(ECB*);
void DloHangup(void);
void DPCPDTerminate(void);
void DPCPDBackground(void);
void DPCFileMain(void* arg);    // thread
}
#include "sfxwatch.h"
#include "sfxqview.h"
#include "sfxparsr.h"

unsigned long GetTickCount(void) {
  return clock() * 1000 / CLOCKS_PER_SEC;
}

extern int DloState;
extern LONG DloPXmitCount;
extern LONG DloPMaxBufferSize;
extern LONG DloRcvCount;
extern LONG DloConn;

int DloGetCurrentState(void) {
#if 1
  return (DloState == DLOS_CONN && DloConn == DLO_CONN_PACKAGE) ? DLOS_CONN : DL
OS_IDLE;
#else
  return DloState;
#endif
}

int DloPortEmpty(void) {
#if 1
  return DloPXmitCount == 0;
#else
  return DloAndCommEmpty();
#endif
}

int DloPortOpen(void) {
  return AIOPortHandle != (-1);
}

int DloGetStatus(tDloStatus* pStatus) {
  if (pStatus == 0)
    return (-1);
  if (!DloPortOpen())
    return (-1);
  pStatus->iState = DloGetCurrentState();
  pStatus->iXmitBytesBuffered = DloPXmitCount;
  pStatus->iXmitBufferSpace = DloPMaxBufferSize;
  pStatus->iRcvBytesBuffered = DloRcvCount;
  pStatus->iRcvBufferSpace = DLOBUFSIZE;
  return 0;
}

int DloGetBufSize(void) {
```

```
  return DloPMaxBufferSize - DloPXmitCount;
}

DWORD DloExtendInactivityTimer(long) {
}

void DloHangup(void) {
}

void DloDispatch(void) {
}

/*
 * Returns whether the adapter can gain access to the passed group ID
 * The group ID includes a version number.
 */
long DLLAPI CDBCheckGroupID(CdbCfg_t *cfg)
{

        if(find_pacau(cfg->groupid, cfg->ver) != NULL)
                return(CAS_IMPLICIT);
        if(find_dacau(cfg->groupid, cfg->ver) != NULL)
                return(CAS_AUTHENTICATED);
        if(find_ecau(cfg->groupid, cfg->ver) != NULL)
                return(CAS_EXPLICIT);
        return(CAS_ERROR);
}

/*
 * Returns a version number which increments when there have been
 * ANY changes to the adapter's conditional access.
 */
long DLLAPI CDBCheckCAChange(void)
{
        return CDBVersion;
}


struct PID {
  PID()         { DPCFilePID = GetThreadID(); }
  ~PID()        { DPCFilePID = 0; }
};

struct Semaphore {
  LONG handle;
  Semaphore(long initial = 0) { handle = OpenLocalSemaphore(initial); }
  ~Semaphore() { if (handle) CloseLocalSemaphore(handle); }
  void Signal(void) { SignalLocalSemaphore(handle); }
  LONG Wait(int milliseconds = (-1)) { return TimedWaitOnLocalSemaphore(handle,
(LONG)milliseconds); }
  LONG value(void) { return ExamineLocalSemaphore(handle); }
  LONG operator --(void);
};

inline LONG Semaphore::operator --(void) {
  LONG v = value();
  if (v == 0)
    return v;
  WaitOnLocalSemaphore(handle);
  return v - 1;
}

Semaphore* DPCPDSemaphore;
SfxDispatcher* pDispatcher;
QUEUEVIEWER* pQueueViewer;
ECBQueue DPCPDQueue;
```

```
int PD_ESR(ECB* ecb) {
  Enqueue_IntsDisabled(&DPCPDQueue, ecb);
  return 0;
}

long BicddSignText(char* p_string,
                   unsigned long size,
                   char* p_sign) {
  return DIOSignText(p_string, size, p_sign);
}

long BicddGetSN(char* p_serial_num) {
  DIOGetSN(p_serial_num);
  return 0;
}

long BicddOpenChannel(BICDD_CHANNEL_CONFIG* channel_config) {
  if (channel_config->num_addresses != 1)
    return 1;

  DPCPDSemaphore = new Semaphore();
  if (DPCPDSemaphore->handle == 0) {
    delete DPCPDSemaphore;
    DPCPDSemaphore = 0;
    return 2;
  }
  DPCPDQueue.semaphore = DPCPDSemaphore->handle;

  // there is actually an overflow here IRT channel being a short!
  long ret = DIOOpenChannel(channel_config->address[0],
                            PD_ESR,
                            (LONG*)&channel_config->channel);
  if (ret) {
    delete DPCPDSemaphore;
    DPCPDSemaphore = 0;
    DPCPDQueue.semaphore = 0;
  }
  return ret;
}

long BicddCloseChannel(unsigned long channel) {
  long ret = DIOCloseChannel(channel);
  if (ret)
    return ret;
  delete DPCPDSemaphore;
  DPCPDSemaphore = 0;
  DPCPDQueue.semaphore = 0;
  while (DPCPDQueue.head) {
    ECB* ecb = Dequeue(&DPCPDQueue);
    CLSLReturnRcvECB(ecb);
  }
  return 0;
}

/*************************************************************************
 *                                                                       *
 *                       ELEMENTS SECTION                                *
 *                   ( Elements Table support )                          *
 *                                                                       *
 *************************************************************************/

CDBelement_t Elements[MAXELEMENTS];

static find_element_by_mac(MACaddr_t mac)
{
```

```
  int k;
  int ret = -1;

  for(k = 0; k < MAXELEMENTS; k++) {
    if(Elements[k].in_use == 'Y' &&
       memcmp(&Elements[k].e_mac, &mac, sizeof(mac)) == 0) {
      ret = k;
      break;
    }
  }
  return ret;
}

static add_element(unsigned long channel, ID id, unsigned char ver,
                   MACaddr_t mac, char pack_feed)
{
  int k, ret = CAS_OK;

  if(find_element_by_mac(mac) != -1)
    return(CAS_DUPLICATE_ADDR);
  for(k = 0; k < MAXELEMENTS; k++)
    if(Elements[k].in_use != 'Y')
      break;
  if(k == MAXELEMENTS)
    ret = CAS_ERROR;
  else {
    Elements[k].channel = channel;
    Elements[k].e_ver = ver;
    memcpy(&Elements[k].e_id, &id, sizeof(id));
    memcpy(&Elements[k].e_mac, &mac, sizeof(mac));
    Elements[k].in_use = 'Y';
    Elements[k].packfeed = pack_feed;
  }
  return ret;
}

static find_element_id(ID id, unsigned char ver)
{
  int k;
  int ret = -1;

  for(k = 0; k < MAXELEMENTS; k++) {
    if(Elements[k].in_use == 'Y' &&
       memcmp(&Elements[k].e_id, &id, sizeof(id)) == 0 &&
       Elements[k].e_ver == ver) {
      ret = k;
      break;
    }
  }
  return ret;
}

static del_element_by_mac(MACaddr_t mac)
{
  int k, ret = CAS_ERROR;

  for(k = 0; k < MAXELEMENTS; k++) {
    if(Elements[k].in_use == 'Y' &&
       memcmp(&Elements[k].e_mac, &mac, sizeof(mac)) == 0) {
      ret = CAS_OK;
      Elements[k].in_use = 'N';
      break;
    }
  }
  return ret;
}
```

```
/***********************************************************************
 * Add / Delete Package Delivery Address
 */

/*
 * Allows an application to request reseption of a single additional DPC MAC
 * address. Caller supplies the address's elementID and version number and the
 * element's group ID and version number. CDB looks up the group key and
 * element key for the address and attempts to add the address via a
 * driver call
 */
long BicddAddPKGAddr(CdbCfg_t* cfg) {
  char e_id_txt[7];
  MUXpacau_t* pacau;
  MUXdacau_t* dacau;

  make_element_id((BYTE*)&cfg->elementid, e_id_txt);
  MACbuildAddr(e_id_txt, MAC_PKG, cfg->ver, &cfg->mac);

  dacau = find_dacau(cfg->groupid, cfg->ver);
  pacau = dacau ? (MUXpacau_t*)dacau : find_pacau(cfg->groupid, cfg->ver);
  if(pacau == NULL)
    return CAS_ERROR;
  if (add_element(cfg->channel, cfg->elementid, cfg->ver, cfg->mac, 'P'))
    return CAS_ERROR;
  if (DIOAddGroupAddress(cfg->channel,
                         (BYTE*)&cfg->mac,
                         (BYTE*)&pacau->g_key) ==
      ESUCCESS)
    return CAS_OK;
  del_element_by_mac(cfg->mac);
  return CAS_ERROR;
}

/*
 * For use by package delivery. Allows an application to request
 * reception of a for-sale package ( a package from an explicit group).
 * Package delivery passes address to be received (including the version number)
 * plus the group key to be used to receive the package. This group key was
 * received via explicit request transaction with the NOC.
 * CDB creates the corresponding element key and calls WBicddAddress.
 */
long BicddAddExpAddr(CdbCfg_t* cfg) {
  if(find_ecau(cfg->groupid, cfg->ver) == 0)
    return CAS_ERROR;

  char e_id_txt[7];
  make_element_id((BYTE*)&cfg->elementid, e_id_txt);
  MACbuildAddr(e_id_txt, MAC_PKG, cfg->ver, &cfg->mac);
  if (add_element(cfg->channel, cfg->elementid, cfg->ver, cfg->mac, 'P'))
    return CAS_ERROR;
  if (DIOAddGroupAddress(cfg->channel,
                         (BYTE*)&cfg->mac,
                         (BYTE*)&cfg->expl_g_key) == ESUCCESS)
    return CAS_OK;
  del_element_by_mac(cfg->mac);
  return CAS_ERROR;
}


/*
 * Allows the application to discontinue reception of a single DPC MAC
 * Package Delivery supplies the element id and version number. CDB merely
 * reformats these values into a DPC MAC address and relays it to WINBICDD.
 */
```

```
long BicddDeletePKGAddr(CdbCfg_t* cfg) {
  int element = find_element_id(cfg->elementid, cfg->ver);
  if (element == (-1))
    return CAS_ERROR;
  if (DIODeleteAddress(cfg->channel, (BYTE*)&Elements[element].e_mac))
    return CAS_ERROR;
  del_element_by_mac(Elements[element].e_mac);
  return CAS_OK;
}

long BicddPoll(unsigned long channel) {
  return DPCPDSemaphore ? DPCPDSemaphore->value() : (-1);
}

long BicddReceive(unsigned long channel,
                  BICDD_BUFFER* p_buffers,
                  unsigned long buf_size,
                  long timeout) {
  if (DPCPDSemaphore == 0)
    return (-1);
  if (DPCPDQueue.head == 0 && DPCPDSemaphore->Wait(timeout) != 0)
    return 0;
  ECB* ecb = DPCPDQueue.head;
  int r = 0;
  int n = buf_size / sizeof(BICDD_BUFFER);
  for (; ecb && n > 0; ecb = ecb->ECB_NextLink, --n) {
    p_buffers->data_size = ecb->ECB_Fragment[0].FragmentLength;
    p_buffers->buf_ptr = ecb->ECB_Fragment[0].FragmentAddress;
    p_buffers->last = 1;
    ++p_buffers;
    ++r;
  }
  return r * sizeof(BICDD_BUFFER);
}

long BicddFreeBuffers(unsigned long channel,
                      BICDD_BUFFER* p_buffers,
                      unsigned long buf_size) {
  int n = buf_size / sizeof(BICDD_BUFFER);
  int i = min(n, DPCPDSemaphore->value());
  for (; n > 0 && DPCPDQueue.head; --n)
    CLSLReturnRcvECB(Dequeue(&DPCPDQueue));
  for (; i > 0; --i)
    --*DPCPDSemaphore;
  return n;
}

long BicddGetSiteID(char* buffer) {
  if (!*SiteID)
    return (-1);
  strncpy(buffer, (char*)SiteID, 9);
  return 0;
}

BOOL BicddGetSatelliteStatus(BICDD_SAT_STATS* Stats, long chan) {
  Stats->MarginalCutoff = MARGINAL_ACQ_VALUE;
  Stats->NormalCutoff = NORMAL_ACQ_VALUE;
  Stats->CurrentValue = DPCGetSignalStrength();
  return TRUE;
}

// The name of the registry/ini key values accessed in this module

static char* pREGKEY_DeleteOnDelivery = "DeleteOnDelivery";
static char* pREGKEY_CooperativeLoading = "CooperativeLoading";
static char* pREGKEY_RebuildOnStartup = "RebuildOnStartup";
static char* pREGKEY_Reconcile = "Reconcile";
static char* pREGKEY_EnableDebug = "EnableDebug";
static char DBS_NAME[] = __FILE__;
static const char magic_key[] = (
 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11
);

/*********************************************************************
*
*       EXPORTED FUNCTION
*
*       DPCCancelDownload(LONG fileID)
*
*       Description:
*               This routine cancels the download of the file associated
*               with the fileID if one is pending. This means closing
*               any open file handles and stoping the modem thread.
*
*       Input:
*               fileID                          - File ID of file to cancel
*
*       Output:
*               nothing
*
*       Returns:
*               0               if download was canceled
*
**********************************************************************/

static struct (
  LONG control;
  LONG ret;
  LONG fileID;
  BOOL cancel;
) crossover;


LONG DPCCancelDownload(LONG fileID)
(
  while (crossover.control)
    delay(100);
  crossover.fileID = fileID;
  crossover.cancel = TRUE;
  crossover.control = GetThreadID();
  while (crossover.control)
    delay(100);

  /* Force the help package status to idle */
  UpdateHelpPortal();

  return crossover.ret;
)

LONG DPCDownloadAFile(LONG fileID)
(
  while (crossover.control)
    delay(100);
  crossover.fileID = fileID;
  crossover.cancel = FALSE;
  crossover.control = GetThreadID();
  while (crossover.control)
    delay(100);

  return crossover.ret;
)
```

```
void DPCPDTerminate(void) (
  pDispatcher->Terminate();
)

void DPCPDBackground(void) (
  PDI_FillList_cross();

  if (crossover.control) (
    LONG fileID = crossover.fileID;
    if (fileID != fsm.getFileid() && fsm.unique(fileID) != SFX_OK)
      crossover.ret = (LONG)(-1);
    else if (crossover.cancel) (
      crossover.ret = fsm.dispatch(fileID, SFXFSM_FILE_NOT_WANTED);
      pDispatcher->CancelLoadingFileID(fileID);
    )
    else if (fsm.isRequestable(fileID)) (
      fsm.dispatch(fileID, SFXFSM_PRECOMMIT);
      crossover.ret =
        fsm.dispatch(fileID,
                     fsm.isForSale(fileID) ? SFXFSM_PURCHASE : SFXFSM_FILE_WANTE
D);
    )
  sendret:
    ResumeThread(crossover.control);
    crossover.control = 0;
  )
)


// this is adapted from sfxdemlp.cpp WinMain and dpcfile.c DPCFileMain

void DPCFileMain(void* arg) (
  PID pid;

  if (!PackageDelivery)
    return;

  DbsProcInit("DPCPD");

  if ((arg && stricmp((char*)arg, "rebuild") == ESUCCESS) ||
      DPCGetProfileInt(PROF_PACKAGEDELIVERY, pREGKEY_RebuildOnStartup, 0)) (
    DPCSetProfileInt(PROF_PACKAGEDELIVERY, pREGKEY_RebuildOnStartup, 0);

    int n = fsm.Rebuild();
    if (n >= 0) (
      DBS_SEND_TRACE1(0, "File database rebuilt with %d entries restored",
    )
    else (
      DBS_SEND_TRACE("File database rebuild failed");
    )

    return;
  )

  // wait until DIOBoard initialized
  while (DIOBoard == 0) (
    if (ExitingFlag)
      return;
    delay(500);
  )

  pDispatcherLro = new SfxDispatcherLro();
  if (!pDispatcherLro) (
    DBS_SEND_ERROR(DBS_FATAL, "Could not construct SfxDispatcherLro");
    goto cleanup;
  }
  pDispatcher = new SfxDispatcher();
  if (!pDispatcher) {
    DBS_SEND_ERROR(DBS_FATAL, "Could not construct SfxDispatcher");
    goto cleanup;
  }
  pQueueViewer = new QUEUEVIEWER(PDI_UpdateDisplay);
  if (!pQueueViewer) {
    DBS_SEND_ERROR(DBS_FATAL, "Could not construct QUEUEVIEWER");
    goto cleanup;
  }

  if (DPCGetProfileInt(PROF_PACKAGEDELIVERY, pREGKEY_Reconcile, 1))
    fsm.ReconcileWith(frd);
  cdb.rebuildDB();

  while (!ExitingFlag) {
    pDispatcher->Run();
    DPCPDBackground();
  }

  pDispatcher->Stop(5000);

  cleanup:
  delete pQueueViewer;
  delete pDispatcher;
  delete pDispatcherLro;
}
```

```
#include "dpcagent.h"           /* Our header file */

/* "fix" conflicting types */
#define AllocateResourceTag __AllocateResourceTag__
#include <advanced.h>
#undef AllocateResourceTag
#include <nwbitops.h>

#define milliclock()    (GetHighResolutionTimer() / 10)

#define activityTimer   ECB_DriverWorkspace.DWs_i32val

/* various flags that control the filter */
#undef  FILTER_DATA_ON_RST
#define WIDEN_TCP_WINDOW
#undef  TCP_ACK_LATENCY /* 10 */
/* if used at all, define "only" 1 of the following */
#undef  DPCInetMaxQueuedBytes   /* 4096 */
#define DPCInetMaxQueuedPackets 64
#if     defined(DPCInetMaxQueuedBytes) && defined(DPCInetMaxQueuedPackets)
#error  Only 1 of DPCInetMaxQueuedBytes and DPCInetMaxQueuedPackets allowed
#endif

/* various flags that control the tunnel */
#undef  TUNNEL_ONLY_TCP

#define IP_VERS(x)              (((BYTE*)x)[0] >> 4)
#define IP_HD_LEN(x)            (((BYTE*)x)[0] & 0x0f)
#define IP_TOS(x)               (((BYTE*)x)[1])
#define IP_TOT_LEN(x)           (((WORD*)x)[1])
#define IP_FLAG_FRAG(x)         (((WORD*)x)[3])
#define IP_PROTO(x)             (((BYTE*)x)[9])
#define IP_CSUM(x)              (((WORD*)x)[5])
#define IP_SRC_ADDR(x)          (((LONG*)x)[3])
#define IP_DST_ADDR(x)          (((LONG*)x)[4])

#define IPPROTO_IPENCAP         0x04

#define UDP_SRC_PORT(x)         (((WORD*)x)[0])
#define UDP_DST_PORT(x)         (((WORD*)x)[1])

#define TCP_SRC_PORT(x)         (((WORD*)x)[0])
#define TCP_DST_PORT(x)         (((WORD*)x)[1])
#define TCP_ACKNUM(x)           (((LONG*)x)[2])
#define TCP_CODE(x)             (((BYTE*)x)[13])
#define TCP_WINDOW(x)           (((WORD*)x)[7])
#define TCP_CSUM(x)             (((WORD*)x)[8])

#define TCP_FIN 0x01
#define TCP_SYN 0x02
#define TCP_RST 0x04
#define TCP_PSH 0x08
#define TCP_ACK 0x10
#define TCP_URG 0x20

ECBQueue TxQ;
ECBQueue NewQ;

struct ResourceTagStructure* TxChainRTag = 0;
struct ResourceTagStructure* TxECBRTag = 0;
LONG TxChainID;
struct ResourceTagStructure* RxChainRTag = 0;
struct ResourceTagStructure* RxECBRTag = 0;
LONG RxChainID;
LONG DPC_IP_Address = 0;
static BYTE ConnectionMask[65536 / 8];
```

```
#ifndef __GNUC__
#define inline
#endif  /* __GNUC__ */

/* ECB Manipulation */

static inline void ReleaseECB(ECB* ecb) {
  if (DIOStats) {
#ifdef  DPCInetMaxQueuedBytes
  if ((DIOStats->QDepth -= ecb->ECB_DataLength) < 0)
    DIOStats->QDepth = 0;
#endif
#ifdef  DPCInetMaxQueuedPackets
    --DIOStats->QDepth;
#endif
  }
  --TxECBRTag->RTResourceCount;
  CLSLFastSendComplete(ecb);
#ifdef LOG_ECB_ACTIVITY
  FastLogMsg(LogECBHandle, (LogClientHandle, LogECBHandle, TRUE,
                            "TINET Release(%08lx)\n", ecb));
#endif /* LOG_ECB_ACTIVITY */
}


inline void Enqueue_IntsDisabled(ECBQueue* q, ECB* ecb) {
  ecb->ECB_NextLink = 0;
  if (q->tail)
    q->tail->ECB_NextLink = ecb;
  ecb->ECB_PreviousLink = q->tail;
  q->tail = ecb;
  if (q->head == 0)
    q->head = ecb;
  SignalLocalSemaphore(q->semaphore);
}


void Enqueue(ECBQueue* q, ECB* ecb) {
  _disable();
  Enqueue_IntsDisabled(q, ecb);
  _enable();
}


ECB* Dequeue(ECBQueue* q) {
  ECB* ecb;
  _disable();
  ecb = q->head;
  if (ecb == 0) {
    _enable();
    return 0;
  }
  q->head = ecb->ECB_NextLink;
  if (q->head == 0)
    q->tail = 0;
  else
    q->head->ECB_PreviousLink = 0;
  _enable();
  ecb->ECB_NextLink = ecb->ECB_PreviousLink = 0;
  return ecb;
}
```

```
void Remove(ECBQueue* q, ECB* ecb) {
  _disable();
  if (ecb->ECB_NextLink)
    ecb->ECB_NextLink->ECB_PreviousLink = ecb->ECB_PreviousLink;
  else
    q->tail = ecb->ECB_PreviousLink;
  if (ecb->ECB_PreviousLink)
    ecb->ECB_PreviousLink->ECB_NextLink = ecb->ECB_NextLink;
  else
    q->head = ecb->ECB_NextLink;
  _enable();
  ecb->ECB_NextLink = ecb->ECB_PreviousLink = 0;
}


LONG InetQueuePacket(ECB* ecb, LONG board, void* chainID) {
  board = board;                          /* not used */
  chainID = chainID;                      /* not used */

  /* only handle IP packets */
  if (*(LONG*)ecb->ECB_ProtocolID != 0 ||
      *(WORD*)(ecb->ECB_ProtocolID + 4) != htons(0x0800))
    return 1;

#ifdef LOG_ECB_ACTIVITY
  if (LogECBHandle) {
    int TGID = SetThreadGroupID(DPC_TGID);
    LogMsg(LogClientHandle, LogECBHandle, FALSE,
           "TINET Enqueue(%081x)\n", ecb);
    SetThreadGroupID(TGID);
  }
#endif /* LOG_ECB_ACTIVITY */
  Enqueue(&NewQ, ecb);
  return 0;
}


LONG InetControl(void) {
  return 0xffffff81;
}


void ClearConnection(WORD port) {
  if (ScanBits(ConnectionMask, port, port+2) == port) {
    BitClear(ConnectionMask, port);
    --DIOStats->TxOKMultipleCollisions;
  }
}


int AllocateConnection(WORD port) {
  if (ScanBits(ConnectionMask, port, port+2) != port) {
    /* see if there is a connection left */
    if (DIOStats->TxOKMultipleCollisions < DPCMaxConnections) {
      /* allocate the new connection */
      BitSet(ConnectionMask, port);
      ++DIOStats->TxOKMultipleCollisions;
      return 1;
    }
    return 0;
  }
  return 1;
}


LONG ConnectionLimiter(ECB* ecb, LONG board, void* chainID) {
```

```
  BYTE* IPHeader = ecb->ECB_Fragment[0].FragmentAddress;
  BYTE* TCPHeader = 0;
  board = board;                          /* not used */
  chainID = chainID;                      /* not used */

  /* only handle IP packets */
  if (*(LONG*)ecb->ECB_ProtocolID != 0 ||
      *(WORD*)(ecb->ECB_ProtocolID + 4) != htons(0x0800))
    return 1;

  /* double check stats, hopefully upper layer is kosher, but */
  if (DIOStats == 0) {
  releaseECB:
    --RxECBRTag->RTResourceCount;
    CLSLFastSendComplete(ecb);
    return 0;
  }

  /* only check TCP packets to our interface */
  if (IP_PROTO(IPHeader) != IPPROTO_TCP ||
      IP_DST_ADDR(IPHeader) != DPC_IP_Address)
    return 1;

  TCPHeader = IPHeader + IP_HD_LEN(IPHeader) * 4;

  if (ecb->ECB_Fragment[0].FragmentLength < ((TCPHeader + 20) - IPHeader))
    return 1;

  if (TCP_CODE(TCPHeader) & (TCP_FIN|TCP_RST)) {
    /* release the connection */
    ClearConnection(ntohs(TCP_DST_PORT(TCPHeader)));
  }
  else if (TCP_CODE(TCPHeader) & TCP_SYN) {
    /* allocate the connection */
    if (!AllocateConnection(ntohs(TCP_DST_PORT(TCPHeader))))
      goto releaseECB;
  }

  return 1;
}


/* IP Manipulation */

#if 0
char *chksum (BYTE *buf, unsigned cnt)
{
        static unsigned char crc_bytes[2];
        BYTE rbl;
        WORD rax, rcx;
        int redx;
        BYTE *rdssi;

        crc_bytes[0] = crc_bytes[1] = 0;

        rcx = cnt;
        rdssi = buf;
        rbl = rcx;
        rcx = rcx >> 1;
        redx = 0;
        if (rcx != 0)
        {
                while (rcx--)
                {
                        rax = *((WORD *)rdssi);
                        rdssi += 2;
                if (redx & 0xffff0000)
                        redx++;
                redx &= 0x0000ffff;
                redx += rax;
        }
        if (redx &= 0x0000ffff)
        {
                redx &= 0x0000ffff;
                redx++;
        }
    }
    if (rbl & 1)
    {
        rax = 0;
        rax = *rdssi;
        redx += rax;
        if`(redx &= 0x0000ffff)
                redx++;
    }
    redx = ~redx;
    crc_bytes[0] = redx & 0xff;
    crc_bytes[1] = (redx >> 8) & 0xff;
    return (char *)crc_bytes;
);

#endif


#ifdef  __GNUC__
/*
 *      This is a version of ip_compute_csum() optimized for IP headers, which
 *      always checksum on 4 octet boundaries.
 *      this version is constructed from various places in the linux and Hughes
 *      sources.
 */

static inline unsigned short ip_fold_1comp_csum(unsigned long sum) (
  unsigned short csum;
  __asm__("movl %w1, %w0\n\t"
          "shrl $16, %1\n\t"
          "addw %w1, %w0\n\t"
          "adcw $0, %w0\n\t"
          "notw %w0"
          : "=a" (csum)
          :  "b" (sum));
  return csum;
)


static inline unsigned short ip_fast_csum(unsigned short * buff, int wlen) (
  unsigned long sum = 0;

  if (wlen) (
    unsigned long eax;
    /* Suggested speedup:
       1:
       movl (%esi), %ebx
       lea (%esi+4), %esi
       adcl %ebx, %eax
       decl %ecx
        jnz 1b
       adcl $0, %eax
       movl %eax, %ebx
       shrl $16, %eax
       addw %ebx, %eax
       adcl $0, %eax
```

```
       xorl $0xffff, %eax
        */
    __asm__("clc\n"
            "1:\t"
            "lodsl\n\t"
            "adcl %3, %0\n\t"
            "loop 1b\n\t"
            "adcl $0, %0\n\t"
            : "=r" (sum), "=S" (buff), "=c" (wlen), "=a" (eax)
            :  "0" (sum),  "1" (buff),  "2" (wlen));
  )
  return ip_fold_1comp_csum(sum);
)

#define chksum(b, 1)    ip_fast_csum(b, (1) / 4)

static inline unsigned short ip_adjust_csum(unsigned short oldcsum,
                                            unsigned short oldval,
                                            unsigned short newval) (
  unsigned long sum = ((unsigned short)~oldcsum);
  sum += ((unsigned short)-oldval);
  sum += newval;
  return ip_fold_1comp_csum(sum);
)

#endif /* __GNUC__ */


static int DummyFrame(FRAG_DESC* frag) (
  frag = frag;                              /* not used */
  return 0;
)


int (*DPCDropFrame)(FRAG_DESC* frag) = DummyFrame;


void FilterQueue(void* arg) (
  ECB* ecb;
  ECB* rover;
  BYTE* IP;
  BYTE* TCP;
  int excess;
  arg = arg;                                /* not used */

  RenameThread(GetThreadID(), "DPCAgent Filter");

  for (;;) (
    if (ExitingFlag)
      return;
    TimedWaitOnLocalSemaphore(NewQ.semaphore, 1000);
    if (!NewQ.head)
      continue;

    ecb = Dequeue(&NewQ);
    ecb->activityTimer = milliclock();
    IP = ecb->ECB_Fragment[0].FragmentAddress;

    if (DIOStats == 0) (
    releaseECB:
      DPCDropFrame((FRAG_DESC*)&ecb->ECB_FragmentCount);
      --TxECBRTag->RTResourceCount;
      CLSLFastSendComplete(ecb);
#ifdef LOG_ECB_ACTIVITY
      FastLogMsg(LogECBHandle, (LogClientHandle, LogECBHandle, TRUE,
                                "TINET Release(%081x)\n", ecb));
```

```
#endif /* LOG_ECB_ACTIVITY */
      continue;
    }

    /* always send a fragmented or routed packet */
    if (ecb->ECB_Fragment[0].FragmentLength < 20 ||
        IP_FLAG_FRAG(IP) & htons(0x3fff) ||
        IP_SRC_ADDR(IP) != DPC_IP_Address) {
    enqueueTxQ:
#ifdef  DPCInetMaxQueuedBytes
      if (DIOStats->QDepth > DPCInetMaxQueuedBytes) {
        ++DIOStats->RetryTxCount;
        goto releaseECB;
      }
      DIOStats->QDepth += ecb->ECB_DataLength;
#endif
#ifdef  DPCInetMaxQueuedPackets
      if (DIOStats->QDepth > DPCInetMaxQueuedPackets) {
        ++DIOStats->RetryTxCount;
        goto releaseECB;
      }
      ++DIOStats->QDepth;
#endif
      Enqueue(&TxQ, ecb);
      continue;
    }

    excess = ecb->ECB_Fragment[0].FragmentLength - IP_HD_LEN(IP) * 4;
    if (excess > 0) {
      TCP = IP + IP_HD_LEN(IP) * 4;
    }
    else {
      TCP = ecb->ECB_Fragment[1].FragmentAddress + (-excess);
      excess += ecb->ECB_Fragment[1].FragmentLength;
    }

    if (IP_PROTO(IP) == IPPROTO_UDP)
      goto filterUDP;

    if (IP_PROTO(IP) != IPPROTO_TCP)
      goto enqueueTxQ;

  filterTCP:
    if (excess < 20)
      goto enqueueTxQ;

    if (TCP_CODE(TCP) & TCP_SYN) {
      if (!AllocateConnection(ntohs(TCP_SRC_PORT(TCP))))
        goto releaseECB;
      /* scan for duplicate in TxQ */
      for (rover = TxQ.head; rover; rover = rover->ECB_NextLink) {
        BYTE* roverIP = rover->ECB_Fragment[0].FragmentAddress;
        BYTE* roverTCP;
        excess = (rover->ECB_Fragment[0].FragmentLength -
                  IP_HD_LEN(roverIP) * 4);
        if (excess > 0) {
          roverTCP = roverIP + IP_HD_LEN(roverIP) * 4;
        }
        else {
          roverTCP = rover->ECB_Fragment[1].FragmentAddress + (-excess);
          excess += rover->ECB_Fragment[1].FragmentLength;
        }
        if (rover->ECB_Fragment[0].FragmentLength >= 20 &&
            excess >= 20 &&
            (IP_FLAG_FRAG(roverIP) & htons(0x3fff)) == 0 &&
            IP_PROTO(roverIP) == IPPROTO_TCP &&
```

```
            TCP_CODE(roverTCP) == TCP_CODE(TCP) &&
            IP_DST_ADDR(roverIP) == IP_DST_ADDR(IP) &&
            TCP_DST_PORT(roverTCP) == TCP_DST_PORT(TCP) &&
            IP_SRC_ADDR(roverIP) == IP_SRC_ADDR(IP) &&
            TCP_SRC_PORT(roverTCP) == TCP_SRC_PORT(TCP)) {
          ++DIOStats->TxOKSingleCollision;
          goto releaseECB;
        }
      }
      goto enqueueTxQ;
    }
#ifdef  FILTER_DATA_ON_RST
    if (TCP_CODE(TCP) & TCP_RST) {
      /* scan for data in TxQ */
      for (rover = TxQ; rover; rover = rover->ECB_NextLink) {
        BYTE* roverIP = rover->ECB_Fragment[0].FragmentAddress;
        BYTE* roverTCP;
        excess = (rover->ECB_Fragment[0].FragmentLength -
                  IP_HD_LEN(roverIP) * 4);
        if (excess > 0) {
          roverTCP = roverIP + IP_HD_LEN(roverIP) * 4;
        }
        else {
          roverTCP = rover->ECB_Fragment[1].FragmentAddress + (-excess);
          excess += rover->ECB_Fragment[1].FragmentLength;
        }
        if (rover->ECB_Fragment[0].FragmentLength >= 20 &&
            excess >= 20 &&
            (IP_FLAG_FRAG(roverIP) & htons(0x3fff)) == 0 &&
            IP_PROTO(roverIP) == IPPROTO_TCP &&
            IP_DST_ADDR(roverIP) == IP_DST_ADDR(IP) &&
            TCP_DST_PORT(roverTCP) == TCP_DST_PORT(TCP) &&
            IP_SRC_ADDR(roverIP) == IP_SRC_ADDR(IP) &&
            TCP_SRC_PORT(roverTCP) == TCP_SRC_PORT(TCP) &&
            TCP_CODE(roverTCP) != TCP_CODE(TCP)) {
          rover->activityTimer = 0;     /* will get taken out shortly */
          ++DIOStats->TxAbortCarrierSense;
        }
      }
      /* fallthru */
    }
#endif
    if (TCP_CODE(TCP) & (TCP_RST|TCP_FIN)) {
      ClearConnection(ntohs(TCP_SRC_PORT(TCP)));
      goto enqueueTxQ;
    }
    if (TCP_CODE(TCP) & TCP_ACK) {
#ifdef WIDEN_TCP_WINDOW
      WORD oldwin = TCP_WINDOW(TCP);
      WORD newwin = ntohs(oldwin);
      if (newwin < 40000) {
        newwin += (newwin >> 1);        /* *= 1.5, will be <= 60K! */
        newwin = htons(newwin);
        TCP_WINDOW(TCP) = newwin;
        TCP_CSUM(TCP) = ip_adjust_csum(TCP_CSUM(TCP),
                                       oldwin,
                                       newwin);
      }
#endif /* WIDEN_TCP_WINDOW */
      if (TCP_CODE(TCP) & (TCP_URG|TCP_PSH))
        goto enqueueTxQ;
#ifdef TCP_ACK_LATENCY
      ecb->activityTimer = milliclock() + TCP_ACK_LATENCY;
#endif
      /* scan for redundancy in TxQ */
      for (rover = TxQ.head; rover; rover = rover->ECB_NextLink) {
```

```
      BYTE* roverIP = rover->ECB_Fragment[0].FragmentAddress;
      BYTE* roverTCP;
      excess = (rover->ECB_Fragment[0].FragmentLength -
                IP_HD_LEN(roverIP) * 4);
      if (excess > 0) {
        roverTCP = roverIP + IP_HD_LEN(roverIP) * 4;
      }
      else {
        roverTCP = rover->ECB_Fragment[1].FragmentAddress + (-excess);
        excess += rover->ECB_Fragment[1].FragmentLength;
      }
      if (rover->ECB_Fragment[0].FragmentLength >= 20 &&
          excess >= 20 &&
          (IP_FLAG_FRAG(roverIP) & htons(0x3fff)) == 0 &&
          IP_PROTO(roverIP) == IPPROTO_TCP &&
          IP_DST_ADDR(roverIP) == IP_DST_ADDR(IP) &&
          TCP_DST_PORT(roverTCP) == TCP_DST_PORT(TCP) &&
          IP_SRC_ADDR(roverIP) == IP_SRC_ADDR(IP) &&
          TCP_SRC_PORT(roverTCP) == TCP_SRC_PORT(TCP) &&
          TCP_CODE(roverTCP) & TCP_ACK &&
          (htonl(TCP_ACKNUM(roverTCP)) + htons(TCP_WINDOW(roverTCP)) <
           htonl(TCP_ACKNUM(TCP)) + htons(TCP_WINDOW(TCP)))) {
        /* move ACK information over to TxQ and release this packet */
        TCP_CSUM(roverTCP) = ip_adjust_csum(TCP_CSUM(roverTCP),
                                            TCP_WINDOW(roverTCP),
                                            TCP_WINDOW(TCP));
        TCP_CSUM(roverTCP) = ip_adjust_csum(TCP_CSUM(roverTCP),
                                            (WORD)TCP_ACKNUM(roverTCP),
                                            (WORD)TCP_ACKNUM(TCP));
        TCP_CSUM(roverTCP) = ip_adjust_csum(TCP_CSUM(roverTCP),
                                            TCP_ACKNUM(roverTCP)>>16,
                                            TCP_ACKNUM(TCP)>>16);
        TCP_ACKNUM(roverTCP) = TCP_ACKNUM(TCP);
        TCP_WINDOW(roverTCP) = TCP_WINDOW(TCP);
        ++DIOStats->TxAbortExcessCollisions;
        goto releaseECB;
      }
    }
    goto enqueueTxQ;
  }
  goto enqueueTxQ;

filterUDP:
  {
    BYTE* UDP = TCP;
    BYTE* DNS;

    /* ECB contents determined by inspection, there are safer methods */
    if (excess < 8)
      goto enqueueTxQ;

    /* filter DNS only */
    if (UDP_DST_PORT(UDP) != htons(53))
      goto enqueueTxQ;

    excess -= 8;
    DNS = (excess > 0) ? (UDP + 8) : ecb->ECB_Fragment[1].FragmentAddress;

    for (rover = TxQ.head; rover; rover = rover->ECB_NextLink) {
      BYTE* roverIP = rover->ECB_Fragment[0].FragmentAddress;
      BYTE* roverUDP;
      BYTE* roverDNS;
      excess = (rover->ECB_Fragment[0].FragmentLength -
                IP_HD_LEN(roverIP) * 4);
      if (excess > 0) {
        roverUDP = roverIP + IP_HD_LEN(roverIP) * 4;
```

```
      }
      else {
        roverUDP = rover->ECB_Fragment[1].FragmentAddress + (-excess);
        excess += rover->ECB_Fragment[1].FragmentLength;
      }
      if (rover->ECB_Fragment[0].FragmentLength >= 20 &&
          excess >= 8 &&
          (IP_FLAG_FRAG(roverIP) & htons(0x3fff)) == 0 &&
          IP_PROTO(roverIP) == IPPROTO_UDP &&
          IP_DST_ADDR(roverIP) == IP_DST_ADDR(IP) &&
          UDP_DST_PORT(roverUDP) == UDP_DST_PORT(UDP) &&
          IP_SRC_ADDR(roverIP) == IP_SRC_ADDR(IP) &&
          UDP_SRC_PORT(roverUDP) == UDP_SRC_PORT(UDP) &&
          (roverDNS = (((excess -= 8) > 0) ?
                       (roverUDP + 8) :
                       rover->ECB_Fragment[1].FragmentAddress)) &&
          *(LONG*)DNS == *(LONG*)roverDNS) {
        ++DIOStats->TxAbortLateCollision;
        goto releaseECB;
      }
    }
    goto enqueueTxQ;
  }
 }
}


/* SLIP, PPP, Modem Manipulation */

#define MAX_READ_BUF 128

int InetState = MODEM_IDLE;
static BYTE SlipEndPkt[1] = {END};


int WaitingLines = 0, NextWait = 0;
char WaitingBuffer[MAX_READ_BUF];
int WaitingIndex = 0;
LONG ConnectingTimeout = 0;
LONG ConnectingRedial = FALSE;

int BaudRate[] =
{
        2400,                           /* 0 */
        3600,                           /* 1 */
        4800,                           /* 2 */
        7200,                           /* 3 */
        9600,                           /* 4 */
        19200,                  /* 5 */
        38400,                  /* 6 */
        57600,                  /* 7 */
        115200                  /* 8 */
};

void InitLogin()
{
        int i;
        char *nextWait;

        WaitingLines = 0;
        if (DioCfg.auto_login)
        {
                WaitingIndex = 0;
                WaitingBuffer[WaitingIndex] = '\0';
                NextWait = 0;
		ConnectingTimeout = 0;
		for (i = 0, nextWait = DloCfg.wait_for_1; i < 9 ; i++, nextWait+
=sizeof(DloCfg.wait_for_1))
		{
			if (*nextWait)
				WaitingLines++;
		}
	}

}


static BYTE MTUBuffer[8192];


int SLIPSendRoutineOpt(FRAG_DESC* fragStruc)
{
	LONG count = 0;
	BYTE* output = MTUBuffer;

	*output++ = END;
	while (count < fragStruc->FragmentCount)
	{
		FRAGMENTSTRUCT* frag = fragStruc->FragmentDesc + count;
		BYTE* frame = (BYTE*)frag->FragmentAddress;
		LONG length = frag->FragmentLength;

		while (length-- > 0)
		{
			switch (*frame)
			{
				case END:
					*output++ = ESC;
					*output++ = ESC_END;
					break;
				case ESC:
					*output++ = ESC;
					*output++ = ESC_ESC;
					break;
				default:
					*output++ = *frame;
					break;
			}
			++frame;
		}
		++count;
	}
	*output++ = END;

	if (output - MTUBuffer < 22 ||
	    output - MTUBuffer > DloGetWriteBufferSpace())
		return 0;		/* oh, well */
	DloSend(MTUBuffer, output - MTUBuffer, DLO_INET_TIMEOUT);
	return 1;
}


int SLIPSendRoutineDebug(FRAG_DESC* fragStruc)
{
	LONG count = 0;
	BYTE* output = MTUBuffer;
	BYTE* dataStart = 0;		/* separate HEADER from PAYLOAD */
	LONG header = 0x80000000;

	while (count < fragStruc->FragmentCount)
	{
```

```
		FRAGMENTSTRUCT* frag = fragStruc->FragmentDesc + count;
		BYTE* frame = (BYTE*)frag->FragmentAddress;
		LONG length = frag->FragmentLength;

		while (length-- > 0)
		{
			switch (*frame)
			{
				case END:
					*output++ = ESC;
					*output++ = ESC_END;
					break;
				case ESC:
					*output++ = ESC;
					*output++ = ESC_ESC;
					break;
				default:
					*output++ = *frame;
					break;
			}
			if (header == 0x80000000)
				header = (*frame & 0x0f) * 4;
			if (--header == 0)
				dataStart = output;
			++frame;
		}
		++count;
	}

	if (output - MTUBuffer < 20 ||
	    output - MTUBuffer > DloGetWriteBufferSpace())
		return 0;		/* oh, well */
	DloSend(SlipEndPkt, 1, DLO_INET_TIMEOUT);
	DloSend(MTUBuffer, dataStart - MTUBuffer, DLO_INET_TIMEOUT);
	DloSend(dataStart, output - dataStart, DLO_INET_TIMEOUT);
	DloSend(SlipEndPkt, 1, DLO_INET_TIMEOUT);
	return 1;
}


int (*DPCTxFrame)(FRAG_DESC* fragStruc) = SLIPSendRoutineOpt;


/***************************************************************************
 *
 *	IPSendRoutine(ECB *tcb)
 *
 *	Description:
 *
 *	Input:
 *		ecb					- Pointer to Eve
nt Control Block
 *
 *	Output:
 *		nothing
 *
 *	Returns:
 *		0 if finished with ECB
 *
 ***************************************************************************/

static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
	0x45,				/* version 4, length 5 */
	0,				/* tos */
	0, 0,				/* length */
```

```
        0, 0,                           /* ident */
        0, 0,                           /* fragment */
        0x7f,                           /* ttl */
        4,                              /* IP in IP (encapsulation) */
);
#define IPHeaderIdent   (*(WORD*)&IPHeader[4])

int     IPSendRoutine(ECB *ecb)
{
        FRAG_DESC* fragStruc = alloca(sizeof(LONG) + (sizeof(FRAGMENTSTRUCT) * (
ecb->ECB_FragmentCount + 2)));
        WORD frame_size = ecb->ECB_DataLength;
        int options_collapsed = 1;
        LONG currFrag = 0;
        BYTE* ecbIPHeader = ecb->ECB_Fragment[0].FragmentAddress;

        /* initialize the copy of the tcb fragStruct */
        memcpy(fragStruc,
               &ecb->ECB_FragmentCount,
               sizeof(LONG) + (sizeof(FRAGMENTSTRUCT) * ecb->ECB_FragmentCount))
;
        if (frame_size < DloCfg.mtu &&
            (
#ifdef  TUNNEL_ONLY_TCP
             /* UDP doesn't need tunnel header */
             (ecbIPHeader[9] != IPPROTO_TCP) ||
#endif
             /* either do "routed" packets */
             (*(LONG*)&ecbIPHeader[12] != DPC_IP_Address)))
               goto skipFragger;

        memset(&fragStruc->FragmentDesc[fragStruc->FragmentCount],
               0,
               sizeof(FRAGMENTSTRUCT) * 2);

        frame_size += IP_TUNNEL_SIZE;
        /*
         * fill IPHeader with tunnel data, including IP/gateway addresses,
         * and prepend to frag list.
         */
        *(WORD*)(&IPHeader[2]) = htons(frame_size);
        ++IPHeaderIdent;
        *(WORD*)(&IPHeader[10]) = 0; /* checksum, for now */
        *(LONG*)(&IPHeader[12]) = DloCfg.ip_address;
        *(LONG*)(&IPHeader[16]) = DloCfg.gateway_address;
        memmove(fragStruc->FragmentDesc + 1,
                fragStruc->FragmentDesc,
                sizeof(FRAGMENTSTRUCT) * fragStruc->FragmentCount);
        fragStruc->FragmentDesc[0].FragmentAddress = IPHeader;
        fragStruc->FragmentDesc[0].FragmentLength = IP_TUNNEL_SIZE;
        ++fragStruc->FragmentCount;
        ++currFrag;
        *(WORD*)(&IPHeader[10]) = chksum((WORD *)IPHeader,
                                         IP_TUNNEL_SIZE);

        while (frame_size > DloCfg.mtu)
        {
                /*
                 * Shucks.  Have to fragment the packet.
                 * This algorithm is roughly per RFC791.
                 */
                LONG OIHL = fragStruc->FragmentDesc[0].FragmentLength;
                BYTE OMF = IPHeader[6] & 0x20;
                LONG NFB   (DloCfg.mtu - OIHL) & 0xfff8;
                WORD TL = OIHL + NFB;
```

```
                IPHeader[6] |= 0x20;    /* set More Fragments */
                *(WORD*)(&IPHeader[2]) = htons(TL);
                *(WORD*)(&IPHeader[10]) = 0; /* clear checksum */
                *(WORD*)(&IPHeader[10]) = chksum((WORD *)IPHeader, OIHL);

                /*
                 * Now fake out the fragStruc to reflect TL.
                 * Hang on to enough information to remove the TL less OIHL
                 * later.
                 */
                TL -= OIHL;
                frame_size -= TL;
                while (TL > 0 &&
                       fragStruc->FragmentDesc[currFrag].FragmentLength <= TL)
                {
                        TL -= fragStruc->FragmentDesc[currFrag].FragmentLength;
                        ++currFrag;
                }
                if (TL > 0)
                {
                        /*
                         * This frag gets split into 2 pieces.
                         */
                        memmove(fragStruc->FragmentDesc + currFrag + 1,
                                fragStruc->FragmentDesc + currFrag,
                                sizeof(FRAGMENTSTRUCT) *
                                (fragStruc->FragmentCount - currFrag));
                        ++fragStruc->FragmentCount;
                        fragStruc->FragmentDesc[currFrag].FragmentLength = TL;
                        ++currFrag;
                        fragStruc->FragmentDesc[currFrag].FragmentLength -= TL;
                        fragStruc->FragmentDesc[currFrag].FragmentAddress = ((ch
ar*)fragStruc->FragmentDesc[currFrag].FragmentAddress) + TL;
                }
                TL = fragStruc->FragmentCount - currFrag + 1;
                fragStruc->FragmentCount = currFrag;

                if (DPCTxFrame(fragStruc) == 0)
                        return 0;

                if (!options_collapsed) {
                        LONG offset = 20;
                        while (offset < OIHL && IPHeader[offset]) {
                                if (IPHeader[offset] & 0x80) /* copy */
                                        offset += IPHeader[offset + 1];
                                else (  /* collapse */
                                        LONG len = IPHeader[offset + 1];
                                        memcpy(IPHeader + offset,
                                               IPHeader + offset + len,
                                               OIHL - (offset + len));
                                        OIHL -= len;
                                }
                        }
                        offset = fragStruc->FragmentDesc[0].FragmentLength;
                        fragStruc->FragmentDesc[0].FragmentLength = (OIHL + 3) &
 0x3c;
                        memset(IPHeader + OIHL,
                               0,
                               fragStruc->FragmentDesc[0].FragmentLength - OIHL)
;
                        IPHeader[0] = 0x40 | (fragStruc->FragmentDesc[0].Fragmen
tLength / 4);

                        frame_size -= offset - fragStruc->FragmentDesc[0].Fragme
ntLength;

                        options_collapsed = 1;
                }
                /*
                 * Adjust the frag list to "remove" the fragment just sent.
                 */
                memmove(fragStruc->FragmentDesc + 1,
                        fragStruc->FragmentDesc + currFrag,
                        sizeof(FRAGMENTSTRUCT) * TL);
                fragStruc->FragmentCount = TL;
                currFrag = 1;

                /* compute new IPHeader values: fragment offset */
                *(WORD*)(&IPHeader[6]) = htons((ntohs(*(WORD*)(&IPHeader[6])) &
                                                0x1fff)
                                              + (NFB / 8));
                IPHeader[6] |= OMF;
                if (frame_size <= DloCfg.mtu)
                {
                        *(WORD*)(&IPHeader[2]) = htons(frame_size);
                        *(WORD*)(&IPHeader[10]) = 0; /* clear checksum */
                        *(WORD*)(&IPHeader[10]) = chksum((WORD *)IPHeader,
                                                 fragStruc->FragmentDesc
[0].FragmentLength);
                        break;
                }
        }

skipFragger:
        /* send the remaining (possibly ALL) portion of the frame */
        if (DPCTxFrame(fragStruc) == 0)
                return 0;

        if (DIOStats)
        {
                ++DIOStats->TotalTxPacketCount;
                if ((DIOStats->TotalTxOKByteCountLow += ecb->ECB_DataLength) <
                    ecb->ECB_DataLength)
                        ++DIOStats->TotalTxOKByteCountHigh; /* wrapped */
        }
        return 1;
}


static void EmptyESR(ECB* ecb) {
}

unsigned char RawEnvelope[14] = {
        0x00, 0x00, 0x0c, 0x0a, 0x0b, 0x0c,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x08, 0x00,
};

ECB RawECB = {
        0,                              /* ECB_NextLink */
        0,                              /* ECB_PreviousLink */
        0,                              /* ECB_Status */
        EmptyESR,                       /* ECB_ESR */
        -1,                             /* ECB_StackID */
        "",                             /* ECB_ProtocolID */
        0,                              /* ECB_BoardNumber */
        "",                             /* ECB_ImmediateAddress */
        { 0 },                          /* ECB_DriverWorkspace */
        { 0 },                          /* ECB_ProtocolWorkspace */
        -1,                             /* ECB_DataLength */
        -1,                             /* ECB_FragmentCount */
        { RawEnvelope, sizeof(RawEnvelope) },
};
```

```
FRAGMENTSTRUCT RawFrags[15] = {         /* technically unsafe, but works */
        IPHeader, sizeof(IPHeader),
};

#define RawSendRoutineDebug RawSendRoutineOpt
int RawSendRoutineOpt(FRAG_DESC* fragStruc) {
  int i;
  for (i = 20000; --i > 0; ) {
    if (RawECB.ECB_Status == 0)
      goto rawSend;
    if (ExitingFlag)
      return 0;
    ThreadSwitchWithDelay();
  }
  return 0;
rawSend:
  i = fragStruc->FragmentCount;
  RawECB.ECB_FragmentCount = i + 1;
  memcpy(RawFrags,
         fragStruc->FragmentDesc,
         i * sizeof(FRAGMENTSTRUCT));
  RawECB.ECB_DataLength = sizeof(RawEnvelope);
  while (i > 0)
    RawECB.ECB_DataLength += RawFrags[--i].FragmentLength;
  CLSLSendPacket(&RawECB);
  return 1;
}

void DisplayWaitStatus(void)
{
        char statusStr[80];
        char *waitingStr;

        if (WaitingLines == 0)
                return;

        waitingStr = (char *)&DloCfg.wait_for_1[NextWait*30];
        NWsprintf(statusStr, MSG("Modem Status: Waiting for \"%s\"\n", 557), wai
tingStr);
        UpdateModemStr(statusStr);
}

void InetStateChange(int state) {
  if (DloCfg.out_protocol == OUT_NETWORK) {
    InetState = PROTOCOL_CONNECTED;
    return;
  }
  switch (state) {
  case DLOS_IDLE:
  case DLOS_DISC_1:
  case DLOS_DISC_2:
  case DLOS_DISC_3:
  case DLOS_DISC_4:
    InetState = MODEM_IDLE;
    if (ConnectingRedial) {
      DloEndConn();
      ConnectingRedial = FALSE;
    }
    break;

  case DLOS_CONN:
    InetState = MODEM_CONNECTED;
    InitLogin();
    if (InetAsleep)
      ResumeThread(DPCInetPID);
    break;
  default:
    break;
  }

}

/***********************************************************************
 * FUNCTION: Convert Internet addess Address
 *
 * DESCRIPTION: converts a character string containing the Internet address
 * into a form that BIC DD understands.
 * e.g. 139.85.124.06 (8B.55.7C.06)  into 067C558B0000
 ***********************************************************************/
void convert_address(char *lpszIpAddress)
{
        char *p;
        int i = 0;
        char tmp[20],tmp1[10];
        tmp[0] = 0;
        while((p=strrchr(lpszIpAddress,(int)'.')) != NULL)
        {
                i = atoi(p+1);
                NWsprintf(tmp1,MSG("%02X", 477),i);
                strcat(tmp,tmp1);
                *p = 0;
        }
        i = atoi(lpszIpAddress);
        NWsprintf(tmp1,MSG("%02X", 478),i);
        strcat(tmp,tmp1);
        strcat(tmp,MSG("0000", 479));
        CStrCpy(lpszIpAddress, tmp);
}

MACaddr_t       HIAddr;
LONG    InetChannel;

void make_hi_key(chunk *key)
{
        int i;
        LONG sn;
        BYTE serialNum[9];
        BYTE serialNumPacked[3];
        BYTE x;

        DIOGetSN(serialNum);
        sn = atol(serialNum);
        NWsprintf(serialNum, MSG("%061x", 480), sn);

        pack_mac_addr(serialNumPacked, 3, serialNum, 6);
        x = serialNumPacked[0];
        serialNumPacked[0] = serialNumPacked[2];
        serialNumPacked[2] = x;

  key->b[0] = serialNumPacked[0] ^ 0xff;
  key->b[1] = serialNumPacked[1] ^ 0xff;
  key->b[2] = serialNumPacked[2] ^ 0xff;
  for(i = 3; i < 8; i++)
        key->b[i] = 0x00 ^ 0xff;

        MACbuildAddr(serialNum, MAC_HI, 0, &HIAddr);
}


void InetChangeProtocol(void)
```

```
{
        switch (DloCfg.out_protocol) {
        case OUT_PPP:
                DPCTxFrame = DebugFlag ? PPPSendRoutineDebug : PPPSendRoutineOpt
;
                break;
        case OUT_NETWORK: {
                void (*ControlEntryPoint)(void) = 0;
                struct DriverConfigurationStructure* dvrCfg = 0;

                if (CLSLGetMLIDControlEntry(DloCfg.net_interface,
                                            &ControlEntryPoint))
                {
                        goto skipDriver;
                }
                dvrCfg = (struct DriverConfigurationStructure *)
                        CommandMlid(DloCfg.net_interface, 0, (LONG)ControlEntryP
oint);
                memcpy(RawEnvelope + 6, dvrCfg->DNodeAddress, 6);

        skipDriver:
                RawECB.ECB_BoardNumber = DloCfg.net_interface;
                memcpy(RawEnvelope, DloCfg.net_addr, 6);

                DPCTxFrame = DebugFlag ? RawSendRoutineDebug : RawSendRoutineOpt
;
                break;
        }
        case OUT_SLIP:
                DPCTxFrame = DebugFlag ? SLIPSendRoutineDebug : SLIPSendRoutineO
pt;
                break;
        }
        InetStateChange(DLOS_DISC_4);
        DloEndConn();
}


int ProcessLogin(void)
{
        BYTE value;
        char *sendStr, *waitStr;
        char sendBuf[40];
        LONG nextTimeout;

        /* No use trying if we aren't even connected */

        /* Get out if we're done */

        if (WaitingLines == 0 || DloCfg.auto_login == FALSE)
        {
                return(TRUE);
        }

        if (!DloConnected())
                return(FALSE);

        /* Timeout if we've waited too long for this wait */

        if (ConnectingTimeout == 0)
        {
                ConnectingTimeout = GetCurrentTime() + DloCfg.wait_timeout_1 * 1
8;
        }

        if (GetCurrentTime() > ConnectingTimeout)
        {
                if (ConnectingRedial == FALSE)
                {
                        /* First timeout. Send return and try again. */
                        ConnectingRedial = TRUE;
                        InitLogin();
                        DloSend(MSG("\r", 181), 1, DLO_INET_TIMEOUT);
                        return(FALSE);
                }
                DloEndConn();
                return(FALSE);
        }

        DisplayWaitStatus();

        while (DloReceive(&value, 1) != 0)
        {
                if (DebugFlag)
                        putchar(value);
                if (value != '\r' && value != '\n')
                {
                        WaitingBuffer[WaitingIndex++] = value;
                        WaitingBuffer[WaitingIndex] = 0;
                        if (WaitingIndex > (MAX_READ_BUF-1))
                                WaitingIndex = 0;
                }

                waitStr = (char *)&DloCfg.wait_for_1[NextWait * 30];
                if(strstr(WaitingBuffer, waitStr) != NULL)
                {
                        sendStr = (char *)&DloCfg.send_1[NextWait * 30];
                        NWsprintf(sendBuf, MSG("%s\r", 558), sendStr);
                        DloSend(sendBuf, CStrLen(sendBuf), DLO_INET_TIMEOUT);
                        NextWait++;
                        WaitingIndex = 0;
                        WaitingBuffer[WaitingIndex] = '\0';
                        WaitingLines--;
                        if (WaitingLines == 0)
                        {
                                DloUpdateModemStr();
                                return(TRUE);
                        }
                        DisplayWaitStatus();
                        nextTimeout = DloCfg.wait_timeout_1 + NextWait;
                        ConnectingTimeout = GetCurrentTime() +
                                        ((nextTimeout) ? (nextTimeout * 18) : (5
*18));
                        return(FALSE);
                } else if (value == '\r')
                {
                        WaitingIndex = 0;
                        WaitingBuffer[WaitingIndex] = '\0';
                }
        }

        return(FALSE);
}

int ConnectProtocol(void)
{
        int ccode;

        if (DloCfg.out_protocol == OUT_SLIP)
        {
                delay(1000);            /* time to "settle" */
                return 1;
```

```
        }
        /*PPP*/
        else
        {
                ccode = ConnectPPP();
        }
        return ccode;
}


void TinetProtocolBind(LONG __parameter) {
  struct EventProtocolBindStruct* epbs =
    (struct EventProtocolBindStruct*)__parameter;
  if (epbs->boardNumber == DIOBoard &&
      epbs->protocolNumber == 1/*PROTOCOL_ID_TCPIP*/) {
    extern LONG DPCNextRegistrationCheck;
    DPCGetIPAddress(&DPC_IP_Address);
    DPCNextRegistrationCheck = 0;
  }
}


/**********************************************************************
*
*       InetMain(void *parm)
*
*       Description:
*               Main thread for Turbo Internet handling.
*
*       Input:
*               parm                                            - ignored
*
*       Output:
*               nothing
*
*       Returns:
*               nothing
*
**********************************************************************/

void InetMain(void *parm)
{
        time_t nextStartConn = 0;
        LONG removedCount = (LONG)(-1);
        long millidelay = 0;
        LONG protocolBindHandle =
          RegisterForEvent(EVENT_PROTOCOL_BIND, TinetProtocolBind, 0);

        parm = parm;                    /* unused */

        NewQ.semaphore = OpenLocalSemaphore(0);
        TxQ.semaphore = OpenLocalSemaphore(0);
        BeginThread(FilterQueue, 0, 0, 0);

        TxChainRTag = AllocateResourceTag(NLMHandle,
                                  MSG("Turbo Inet TxPreScan Chain", 476)
,
                                  LSLTxPreScanStackSignature);
        TxECBRTag = AllocateResourceTag(NLMHandle,
                                 MSG("Turbo Inet Transmit Packets", 619),
                                 ECBSignature);
        RxChainRTag = AllocateResourceTag(NLMHandle,
                                  "Turbo Inet RxPreScan Chain",
                                  LSLPreScanStackSignature);
        RxECBRTag = AllocateResourceTag(NLMHandle,
                                 MSG("Turbo Inet Receive Packets", 619),
                                      ECBSignature);
        DPCGetIPAddress(&DPC_IP_Address);

        if (DloCfg.out_protocol == OUT_NETWORK)
          InetState = PROTOCOL_CONNECTED;

mainloop:
        while (!ExitingFlag)
        {
                InetAsleep = TRUE;
                if (millidelay > 55)
                  delay(millidelay);
                else if (millidelay > 0)
                  ThreadSwitchWithDelay/*LowPriority*/();
                else
                  ThreadSwitch();
                InetAsleep = FALSE;

                while (DIOBoard && removedCount != DIORemovedCount)
                {
                        BYTE address[8];
                        BYTE szBicBCDAddress[20];
                        struct DriverStatsStructure* stats = 0;
                        LONG ip_address = ntohl(DloCfg.ip_address);

                        removedCount = DIORemovedCount;
                        /* Enable internet reception */

                        /* Yuk. We'll change this later to get rid these extra s
tages */
                        NWsprintf(szBicBCDAddress, MSG("%d.%d.%d.%d", 620),
                                  (ip_address >> 24) & 0xff,
                                  (ip_address >> 16) & 0xff,
                                  (ip_address >> 8) & 0xff,
                                  (ip_address) & 0xff);

                        convert_address(szBicBCDAddress);
                        if (!pack_mac_addr(address, 6,
                                           szBicBCDAddress, CStrLen(szBicBCDAddr
ess)))
                        {
                                /* UpdateModemStr(MSG("ERROR: could not pack mac
 address      \n", xxxx)); */
                                millidelay = 500;
                                break;
                        }

                        /* Sending an esr address of -1 tells MLID to handle rec
eption */
                        if (DIOOpenChannel(address,
                                           (int (*)())0xffffffff,
                                           &InetChannel))
                        {
                                millidelay = 500;
                                removedCount = (LONG)(-1);
                                break;
                        }
                        if (ExitingFlag)
                                break;
                        DIOAddHIAddr(InetChannel, (BYTE *)&HIAddr);
                        DPCGetMLIDStats(&stats);
                        DIOStats->TxOKMultipleCollisions = 0;
                        if (CLSLRegisterPreScanTxChain(TxChainRTag,
                                                       DIOBoard,
                                                       3, /* next to last */
                                                       &TxChainID,
```

```
                                                       InetQueuePacket,
                                                       InetControl,
                                                       TxECBRTag))
                        {
                                millidelay = 500;
                                removedCount = (LONG)(-1);
                                break;
                        }
                        if (CLSLRegisterPreScanRxChain(RxChainRTag,
                                                       DIOBoard,
                                                       3, /* next to last */
                                                       &RxChainID,
                                                       ConnectionLimiter,
                                                       InetControl,
                                                       RxECBRTag))
                        {
                                millidelay = 500;
                                removedCount = (LONG)(-1);
                                break;
                        }
                }

                if (DloCfg.out_protocol == OUT_PPP &&
                    InetState == PROTOCOL_CONNECTED)
                {
                        PPPBackground();
                }

                if (TxQ.head == 0 &&
                    (InetState <= MODEM_CONNECTING ||
                     InetState >= PROTOCOL_CONNECTED))
                {
                  TimedWaitOnLocalSemaphore(TxQ.semaphore, 200);
                  millidelay = 0;
                  continue;
                }

                switch (InetState)
                {
                case MODEM_CONNECTED:
                        if (!ProcessLogin())
                        {
                                millidelay = 500;
                                break;
                        }
                        InetState = LOGIN_CONNECTED;
                        ;                       /* fallthru */
                case LOGIN_CONNECTED:
                        if (!ConnectProtocol())
                        {
                                DloEndConn();
                                millidelay = 15 * 1000;
                                break;
                        }
                        InetState = PROTOCOL_CONNECTED;
                        millidelay = 1;
                        break;
                case PROTOCOL_CONNECTED:
                {
                        LONG count = 0;
                        if (DloCfg.out_protocol != OUT_NETWORK &&
                            AIOWriteStatus(AIOPortHandle, &count, 0))
                        {
                                millidelay = 200;
                                break;
                        }
                        if (count == 0)
                        {
                                LONG milliclock = milliclock();
                                ECB* ecb;
                                ecb = TxQ.head;
                                millidelay = 100;
                                while (ecb->activityTimer > milliclock)
                                {
                                        LONG diff = ecb->activityTimer - millicl
ock;
                                        if (diff < millidelay)
                                                millidelay = diff;
                                        if ((ecb = ecb->ECB_NextLink) == 0)
                                                goto mainloop;
                                }
                                Remove(&TxQ, ecb);
                                if (ecb->activityTimer < milliclock() - 60000) {
                                  if (ecb->activityTimer)
                                    ++DIOStats->TxAbortExDeferral;
                                }
                                else
                                  IPSendRoutine(ecb);
                                ReleaseECB(ecb);
                                if (DloCfg.out_protocol != OUT_NETWORK)
                                  AIOWriteStatus(AIOPortHandle, &count, 0);
                        }
                        millidelay = (count *
                                      10 *      /* Tx bits with framing */
                                      1000 /    /* milliseconds */
                                      BaudRate[DloCfg.tinet_baud_index]);
                        break;
                }
                case MODEM_IDLE:
                        if (nextStartConn < time(0))
                        {
                                InitLogin();
                                DloStartConn(DLO_INET_TIMEOUT);
                                InetState = MODEM_CONNECTING;
                                nextStartConn = time(0) + 30;
                                ;               /* fallthru */
                        }
                default:
                        millidelay = 10 * 1000;
                        break;
                }
        }

        DIOCloseChannel(InetChannel);

        CLSLDeRegisterPreScanRxChain(RxChainID);
        CLSLDeRegisterPreScanTxChain(TxChainID);
        while (TxQ.head)
          ReleaseECB(Dequeue(&TxQ));
        while (NewQ.head)
          ReleaseECB(Dequeue(&NewQ));
        CloseLocalSemaphore(TxQ.semaphore); TxQ.semaphore = 0;
        CloseLocalSemaphore(NewQ.semaphore); NewQ.semaphore = 0;

        UnregisterForEvent(protocolBindHandle);

        DPCInetPID = 0;
        return;

}
```